A Fully Analog Memristor-Based Neural Network with Online Gradient Training

Eyal Rosenthal, Sergey Greshnikov, Daniel Soudry*, and Shahar Kvatinsky
Department of Electrical Engineering, Technion, Israel Institute of Technology, Haifa, Israel 3200003
** Department of Statistics, Columbia University, New York, NY 10027, USA*

***Abstract* — In recent years, Neural Networks (NNs) have become widely popular for the execution of different machine learning algorithms. Training an NN is computationally intensive since it requires numerous multiplications of matrices that represent synaptic weights. It is therefore appealing to build a hardware-based NN accelerator to gain parallelism and efficient computation. Recently, we have proposed a compact circuit of a non-volatile synaptic weight based on two CMOS transistors and a memristor. In this paper, we present a fully analog NN design based on our previously proposed synapse with a full design of the different layers and their supporting CMOS circuits. We show that the presented NN significantly reduces the area as compared to a CMOS-based NN, while executing online gradient training with similar accuracy and computational speed improvement as a software implementation.**

Keywords—Multilayer Neural Networks, machine learning, backpropagation, neuromorphic, memristor, CMOS, RRAM

I. INTRODUCTION

Machine learning algorithms have become widely common in computer science for classification problems. One of the most popular machine learning systems is neural networks (NNs). NN algorithms can accurately solve complex tasks with a wide range of input data. For example, a single layer neural network (SNN) consists of input and output neurons, and synapses. The synapses are the connections between the neurons and they represent the weights of the NN. For a multilayer neural network (MNN), the output neurons of the previous layer become the input neurons of the next layer. A common method to train NNs is online gradient descent [1], where the error from the output neurons propagates backwards to the input neurons, and the network updates its synapses to minimize the error.

Realistic datasets require a very large NNs. Training such networks can be prohibitive in both time (*e.g.*, weeks [2] or more) and energy. For example, an NN layer with M inputs and N outputs requires $N \cdot M$ synapses and a single learning step using the online gradient descent method requires $O(N \cdot M)$ operations for updating all the synapses. If the NN is implemented in software, the synapses are represented by matrices. Then the primary computational bottlenecks are a matrix-vector multiplication and a vector-vector outer product [3, 13].

A hardware implementation of an NN can potentially relieve the computational bottleneck due to its massive parallelization nature. The key for hardware implementation is to create weight units that can locally compute, store and update their own weights simultaneously and independently of the other synapses. Previously proposed dedicated hardware solutions have required numerous transistors per synaptic weight [4-11] which resulted in high area and power consumption. Additionally, creating a non-volatile weight unit is difficult with standard CMOS technology.

In this paper, we design a hardware solution that is based on emerging non-volatile memristive technology [12] to store the synaptic weight as proposed in [13]. This solution greatly decreases the number of transistors per synapse as compared to CMOS-only designs. In [13], the NN has been designed and evaluated with ideal periphery circuits without considering their influence on the NN. In this paper, we integrate the synapse design into a fully analog NN circuit, discuss the relevant issues to make it completely functional, and show that the proposed design has high noise robustness and achieves similar accuracy as software execution on machine learning algorithms when comparing it to software execution results.

The rest of the paper is organized as follows. Background on memristors, machine learning and the synapse design is given in Section II. The proposed single layer design is presented in Section III, followed by a multilayer design in Section IV. Results for three different benchmarks are shown in Section V. The paper is concluded in Section VI.

II. BACKGROUND

A. Memristor

Memristors are analog passive devices with varying resistance [12]. Memristors are non-volatile by their definition, and today most of resistive non-volatile memory technologies can be considered as memristors (in their broader definition). Memristors are usually fabricated in a BEOL CMOS process in a metal-insulator-metal structure, resulting in a smaller area than a CMOS transistor. The dynamic behavior of memristors is usually highly non-linear. To accurately model the memristive non-linear behavior in our evaluation, the TEAM model [14] is used in this paper.

B. Online Gradient Descent

The circuit is designed to execute the Online Gradient Descent algorithm, as shown in Fig. 1. To update the weights of the system, the output vector $\boldsymbol{r}$ is first determined by [13]

$$\boldsymbol{r} = \boldsymbol{W}\boldsymbol{x}, \tag{1}$$

978-1-4799-5341-7/16/$31.00 ©2016 IEEE

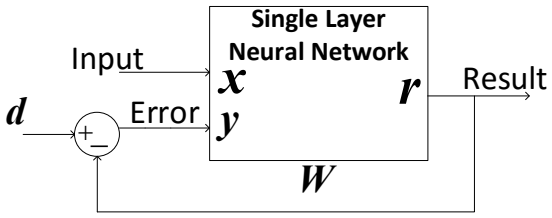

Figure 1: Illustration of an online gradient descent training of a single layer. The system receives an input vector x. The result vector r is subtracted from the desired output vector d. the result of the subtraction y is the error input to the system.

where $\boldsymbol{W}$ is the synaptic weights matrix of the size $N \cdot M$. The synaptic weight update rule [13] is

$$\Delta W_{nm} = \eta x_m y_n, \tag{2}$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are as defined in Fig. 1, $\Delta\boldsymbol{W}$ is the difference of the weight between time steps and η is the learning rate. From (2), the update rule for the synaptic weights is independent for each synapse, thus full parallelism of the training is possible.

C. Synapse Model

In [13], we have proposed a synapse circuit which consists of a PMOS transistor, an NMOS transistor and a memristor, as shown in Fig. 2. The output of the synapse is the current flowing through the memristor. The inputs u and $\overline{u}$ always hold $u = -\overline{u}$. An enable signal e determines which transistor is conducting and consequently the sign of the input sample to the synapse. The enable signal can have three values: $e = V_{DD}$, where only the NMOS is conducting, $e = -V_{DD}$, where only the PMOS is conducting, and $e = 0$, where both transistors are non-conducting. The weight of the synapse W depends on the internal state of the memristor state s and its resultant resistance $R_{mem}(s)$.

To update the synaptic weight based on the sign of the error y, e selects the relevant transistor to enable current in the proper direction through the memristor. The duration of the applied signal e is determined by the magnitude of y. To properly execute (2), x is the voltage across the memristor, η is the rate at which the resistance of the memristor changes. The TEAM model [14] describes a non-linear resistance change, where the magnitude of the memristor current defines the change rate of its state variable, consequently its resistance. To achieve an effective linearity approximation for the memristor behavior around a certain operating point, small update time intervals are used.. Therefore, η depends on the memristor's current, which is defined almost entirely by the magnitude of the input x. For a high voltage input, the training rate is faster than a low voltage input. To read from the synapse, e selects the PMOS transistor, allowing u to be applied across the memristor.

III. Single Layer Design

An SNN layer consists of a synaptic grid and the online gradient descent supporting circuitry. The synaptic grid is based on our previously proposed synapse design [13] with a few changes in the synapse cell and the array as explained in this section. To achieve non-destructive read, the input of the array is normalized by a factor of a to u/a, resulting in current lower than the threshold of the memristor. The output of the synapse is amplified by a factor of a for proper behavior. Figure 3 illustrates a schematic of a two by two synaptic grid.

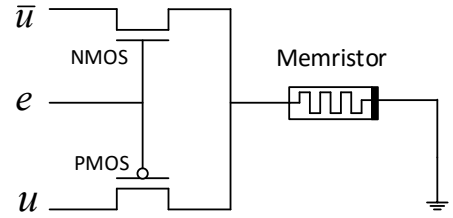

Figure 2: Schematic of the synapse proposed in [13]. The synapse consists of two CMOS transistors and a memristor.

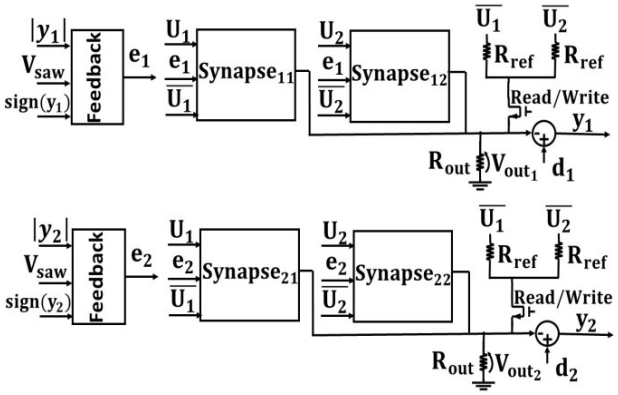

Figure 3: Illustration of a 2x2 single layer showcasing the synaptic grid, the generation of the error y and the feedback circuit which executes online gradient descent.

A. Shock Capacitor

In a fully analog design of the synaptic grid, high voltage spikes occur during the training process. To eliminate these spikes, a relatively small capacitor that absorbs the spikes is added between the memristor and the transistors.

B. Voltage Output

Since the operation during reads requires relatively low operating voltages, it is difficult to use current mirrors to support the use of current as the output. Hence, a resistor R_{out} is added to the output row to convert the output current into voltage. To keep R_{out} from interfering with the read process, the resistor must be significantly low (*i.e.*, $R_{out} \ll R_{mem}(s)$).

C. Subtracting the Operating Point

We define the synaptic weight of a single synapse as

$$W_{nm} = \frac{Vout,n_m}{u_m} = \frac{\frac{u_m}{Rmem(s_{nm})} \cdot Rout}{u_m} = \frac{Rout}{Rmem(s_{nm})}. \tag{3}$$

Where $Vout, n_m$ is the voltage contribution of the n, m synapse. Note that W cannot be zero or negative since s varies between [0…1]. To achieve non-positive synaptic weight values, we define $s = 0.5$ as $W = 0$ by adding a resistor R_{ref}, connected to the input $\bar{u}$ at the output row output, where $R_{ref} = R_{mem}(s = 0.5)$. The synaptic weight of each synapse is then defined as

$$W_{nm} = \left(\frac{1}{Rmem(s_{nm})} - \frac{1}{Rref}\right) \cdot Rout. \tag{4}$$

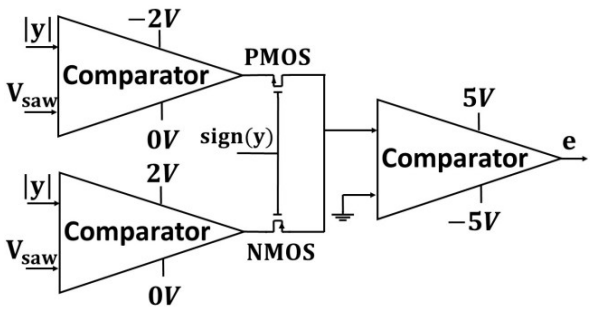

Figure 4: The feedback circuit.

By superposition and the $R_{out} \ll R_{mem}(s)$ constraint, it can be shown that

$$v_{out,n} = \sum_{i=1}^{m} u_i \left(\frac{1}{Rmem(s_{ni})} - \frac{1}{Rref}\right) \cdot Rout, \quad (5)$$

which executes (1). Additionally, a transistor is added to disable the subtraction during the write process. An example is shown in Fig. 3.

D. Feedback Circuit

The feedback circuit, which is the main component for the learning algorithm execution, is shown in Figure 4. The circuit produces a pulse for a time period that is linearly dependent on the error y. The inputs of the feedback circuit are the absolute value of the row error $|y|$, its sign $Sign(y)$ and a sawtooth voltage source V_{saw}. The output of the circuit is a pulse e that defines the duration and direction of the write operation. $Sign(y)$ determines which part of the circuit is active. For $Sign(y) > 0$, the NMOS of Figure 4 is conducting and V_{saw} starts increasing over time. As long as $|y| > V_{saw}$, the comparator outputs its supply positive voltage ($2V$), and the output of the circuit is $e = 5V$. When $V_{saw} \geq |y|$, the comparator outputs 0V and the output of the circuit is $e = 0V$. The operation for $Sign(y) < 0$ is symmetric. The behavior of the signals during training is illustrated in Figure 5.

IV. MULTI LAYER DESIGN

In an MNN, multiple layers of neurons are connected between different synaptic grids, as illustrated in Figure 6. Each neuron layer generates the inputs for the next layer such as (where L is the number of layers)

$$\boldsymbol{x}_{k+1} = \sigma(\boldsymbol{r}_k) \ , k = 1 \dots L-1, \quad (6)$$

where $\sigma(\cdot)$ is a sigmoid activation function. The error of the last layer is determined as in Fig. 1 while the errors for the other layers are determined by

$$\boldsymbol{y}_k = \sigma'(\boldsymbol{r}_k)\boldsymbol{\delta}_{k+1}, \quad (7)$$

where the vector $\boldsymbol{\delta}$ is

$$\boldsymbol{\delta}_{k+1} = \boldsymbol{y}_{k+1} \cdot (\boldsymbol{W}_{k+1})^T. \quad (8)$$

To support MNN execution, additional circuits to mediate between the grids are required. Additionally, there is a third execution step during each training iteration that we call an *inverted read*. Inverted read occurs after the read phase of all the synaptic grids except the first layer. During an inverted read, the roles of the rows and columns of the grid are inverted, thus implementing the transpose weight matrix $\boldsymbol{W}^T$. During the time the grid receives $\boldsymbol{y}$ as input, (8) is implemented.

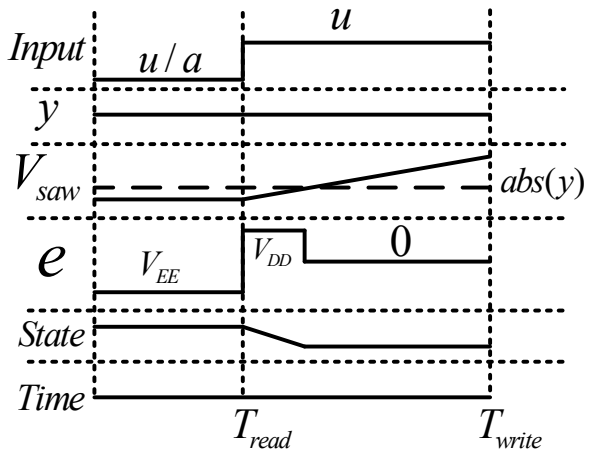

Figure 5: Waveform illustration of a single training iteration.

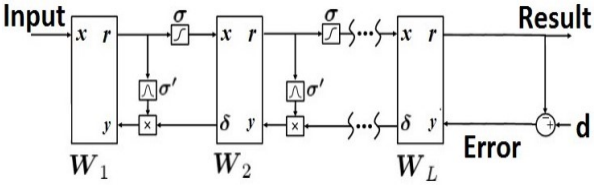

Figure 6: Illustration of a MNN depicting several synaptic grids connected by sigma functions and the generation of the error vector *y* for each grid.

A. Sigma Function

A comparator-based operational amplifier with a relatively small gain is used to implement a simple sigmoid function as the activation function $\sigma(\cdot)$. The output of the comparator can be either 1 or -1, and due to its poor gain, the transition between the different outputs behaves as a sigmoid.

The derivative $\sigma'(\cdot)$ is designed to execute the approximate expression

$$\sigma'(x) \cong \sigma'(0) \cdot (1 - \sigma^2(x)), \quad (9)$$

where $\sigma'(0)$ is the gain of the comparator. Operational amplifiers are used for the multipliers and subtractors. The approximation in (9) is sufficiently accurate – its mean square difference from the numerical derivative is approximately 10^{-11}, for $\sigma'(0) = 500$.

V. EVALUATION

We design the proposed circuit using CMOS 0.18um process and the TEAM model. The circuit parameters are listed in Table 1. For preprocessing we used Zscore Normalization and then Sigmoid Transform [15] on the dataset. All weights have been initialized to W=0. Additionally, a bias synapse has been added to each row with a constant input $V_{bias} = u = -\bar{u}$, the bias synapse acts as an additional weight unit and allows shifting the values of the outputs and activation function. In addition to the normal simulations, noisy simulations have been performed adding up to 10% error to the inputs and simulating thermal noise at 90℃. We compare the results of the analog simulations to Matlab software simulations, implemented as described in Section II. The learning rate of the Matlab simulation is set as $\eta = 0.1$.

To evaluate the accuracy and performance of the proposed design, three datasets [16, 17] have been tested on different networks. Table 2 lists the specifications of the datasets. The

Parameter	Value	Parameter	Value
Power Source		**Memristor**	
V_{DD}, V_{EE}	±5V	$K_{on/off}$	$\pm 100n\frac{m}{s}$
PMOS		$I_{on/off}$	$\pm 1\mu A$
W/L	2.39	$\alpha_{on/off}$	2
NMOS		R_{on}	100Ω
W/L	9.6	R_{off}	200kΩ
Circuit		**Timing**	
V_{saw}	15mV	T_{read}	$5\mu s$
$\sigma'(0)$	500	$T_{inverted-read}$	$5\mu s$
d	8mV	T_{write}	$10\mu s$
a	10	**Bias input**	
R_{out}	1kΩ	V_{bias}	0.18V
R_{ref}	100.05kΩ		

Table 1: Circuit parameters for the analog simulations.

Dataset	Unique training samples	Unique test samples	No. of inputs	No. of outputs	NN size
Wisconsin Diagnostic Breasts Cancer	300	120	30	2	30x2
Wine	96	48	13	3	13x3
Iris	90	60	4	3	4x3

Table 2: Information regarding each dataset and the NN design. Both the Wine and Cancer Diagnostic datasets are linearly separable between all classes, while the Iris dataset is not linearly separable between two of its classes, making it more difficult to be learned.

		Error %			Runtime	
Dataset	***Samples***	***Analog model***	***Noisy analog model***	***Matlab model***	***Analog model***	***Matlab model***
SNN						
Wine	1200	3.75% ±0.52%	2.5% ±0.52%	2.29% ±1.09%	18ms	278.5ms
Breast Cancer	1200	3% ±0.5%	4.67% ±0.67%	3.10% ±1.83%	18ms	210ms
Iris	1080	15.67% ±0.79%	16.5% ±0.67%	15.33% ±0.03%	16.2ms	95.3ms

Table 3: Results of the SNN training simulations of both the analog circuit and the Matlab model. Note that our results differ from the results in [13] since a different error function has been used. For the sake of simplicity, the error function in this work is mean square error, while in [13] a cross entropy error is used.

results of the three datasets are listed in Table 3. For the SNN we get similar training errors as in software with a speed improvement of an order of magnitude. MNN results are not included since they require further optimization of circuit parameters that is planned as a future work.

VI. CONCLUSIONS

The execution of SNN and MNN in hardware has a great potential to surpass the perfomance and energy efficiency of software execution. The proposed use of compact hybrid memristor-CMOS synapses enables efficient hardware for SNN and MNN with online learning. While the previously-presented design of hybrid memristor-CMOS SNN and MNN has achieved impressive accuracy for different machine learning benchmarks, the supporting circuitry has relied on ideal devices and only in this paper the implications of a full analog design are considered.

We show that with a few modifications, such as adding shock capacitors and transferring the output current into voltage, the proposed technique achieves sufficient accuracy. The proposed circuit has significantly lower area than CMOS-based synapses even with the additional devices. The circuit is also an order of magnitude faster than software implementation with the potential for even faster execution. We believe that further investigation of the different implications and parameters of the circuit will further improve the accuracy and performance of the proposed NNs.

ACKNOWLEDGEMENTS

The authors would like to thank Goel Samuel for his technical support. This research is partially supported by the Gruss Lipper Charitable Foundation, Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI) and by the Viterbi Fellowship in the Technion Computer Engineering Center.

REFERENCES

[1] A. Blum, "On-Line Algorithms in Machine Learning," *Online Algorithms: the state of the art*, A. Fiat and G. J. Woeginger (Ed.), Springer-Verlag Berlin Heidelberg, Chapter 14, pp. 306-325, 1998.

[2] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv preprint arXiv:1409.1556*, September 2014.

[3] Y. A. Lecun, L Bottou, G. B. Orr, and K. R. Muller, "Efficent BackProp," *Neural Networks: Tricks of the trade*, 2nd Edition, Springer-Verlag Berlin Heidelberg, Chapter 3, pp. 9-48, 1998.

[4] G. Cauwenberghs, C. F. Neugebauer, and A. Yariv, "Analysis and Verification of an Analog VLSI Incremental Outer-Product Learning System," in *IEEE Transactions on Neural Networks,* Vol. 3, No. 3, pp. 488-497, May 1992.

[5] H. C. Card , C. R. Schneider, and W. R. Moore, "Hebbian Plasticity in MOS Synapses," *IEE Proceedings-F (Radar and Signal Processing)*, Vol. 138, No. 1, pp. 13, February 1991.

[6] C. Schneider and H. Card, "Analogue CMOS Hebbian Synapses," *Electronics letters*, Vol. 29, No. 9, pp. 785-786, April 1991.

[7] M. Valle, D. D. Caviglia, and G. M. Bisio, "An Experimental Analog VLSI Neural Network with On-Chip Back-Propagation Learning," *Analog Integrated Circuits and Signal Processing*, Vol. 9, No. 3, pp. 231-245, April 1996.

[8] C. Lu, B. Shi, and L. Chen, "An On-Chip BP Learning Neural Network with Ideal Nneuron Characteristics and Learning Rate Adaptation," *Analog Integrated Circuits and Signal Processing*, Vol. 31, No. 1, pp. 55-62, April 2002.

[9] T. Morie and Y. Amemiya, "An All-Analog Expandable Neural Network LSI with On-Chip Backpropagation Learning," *IEEE Journal of Solid-State Circuits*, Vol. 29, No. 9, pp. 1086-1093, September 1994.

[10] T. Shima and T. Kimura, "Neuro Chips with On-Chip Back-Propagation and/or Hebbian Learning," *IEEE Journal of Solid-State Circuits*, Vol. 27, No. 12, pp. 1868-1876, December 1992.

[11] T. Simonite, "IBM Chip Processes Data Similar to the Way Your Brain Does," *MIT Techonlogy Report*, August 2014.

[12] L. O. Chua, "Memristor - the Missing Circuit Element," *IEEE Transactions on Circuit Theory*, Vol. 18, No. 5, pp. 507–519, September 1971.

[13] D. Soudry, D. Di Castro, A. Gal, A. Kolodny, and S. Kvatinsky, "Memristor-Based Multilayer Neural Networks With Online Gradient Descent Training," *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 26, No.10, pp. 2408-2421, October 2015.

[14] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "TEAM: ThrEshold Adaptive Memristor Model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 60, No. 1, pp. 211–221, January 2013.

[15] H. Li, C. L. P. Chen and H. P. Huang, "Data Preprocessing," *Fuzzy Neural Intelligent Systems: Mathematical Foundation and the Applications in Engineering, 1st Edition,* CRC Press, Chapter 15, pp. 255-267, September 2000.

[16] M. Lichman, UCI Machine Learning Repository [http://archive.ics.uci.edu/ml], 2013.

[17] O. L. Mangasarian and W. H. Wolberg, "Cancer Diagnosis via Linear Programming," *SIAM News*, Vol. 23, No. 5, pp 1-18, September 1990.