IRWIN AND JOAN JACOBS
CENTER FOR COMMUNICATION AND INFORMATION TECHNOLOGIES

Hebbian Learning Rules with Memristors

Daniel Soudry, Dotan Di Castro, Asaf Gal, Avinoam Kolodny, and Shahar Kvatinsky

CCIT Report #840
September 2013

Electronics
Computers
Communications

DEPARTMENT OF ELECTRICAL ENGINEERING
TECHNION - ISRAEL INSTITUTE OF TECHNOLOGY, HAIFA 32000, ISRAEL

Hebbian Learning Rules with Memristors

Daniel Soudry, Dotan Di Castro, Asaf Gal, Avinoam Kolodny, and Shahar Kvatinsky

***Abstract*—Machine learning algorithms often rely on continuous updating of large matrices of "synaptic weights" by local "Hebbian" rules. These rules generally involve a multiplication term, which poses a challenge for implementing large scale hardware for machine learning. In this paper, a method for performing these multiplications using memristor-based arrays is proposed, based on the fact that approximately, given a voltage pulse, the conductivity of a memristor will increment proportionally to the pulse duration *multiplied by* the pulse magnitude, if the increment is sufficiently small. The proposed method uses a synaptic circuit composed of a small number of components per synapse: one memristor and two CMOS transistors. This circuit is expected to consume between 2% to 8% of the power and area of previous CMOS-only hardware alternative. Such a circuit can be used to implement efficiently scalable machine learning algorithms based on online gradient descent. The utility and robustness of the proposed memristor-based circuit is demonstrated in the standard supervised learning task of handwritten digits recognition.**

***Index Terms*—Memristor, Memristive systems, Machine learning, Adaptive filter, Perceptron, Synapse.**

I. Introduction

ENGINEERING truly intelligent machines is one of the exciting and challenging frontiers of modern hardware design. The field of machine learning (ML) has achieved remarkable progress in the mathematical formulation of performance bounds and algorithms to many classes of problems such as pattern recognition [1–3], natural language processing [4, 5], and time series prediction [6]. ML algorithms have been recently incorporated into numerous commercial products and services such as mobile devices and cloud computing. For realistic tasks, such algorithms perform significantly better when massive computational power is available (*e.g.*, [7–9], and see [10] for related press). This computational intensity has limited their usability in large scale applications, due to area and power requirements. New ML-oriented hardware design approach must therefore be developed to overcome these limitations. In fact, it was recently suggested that such new types of specialized hardware are essential for real progress towards building intelligent machines [11].

Many ML algorithms utilize large matrices of values termed *synaptic weights*. These matrices are continuously updated during the operation of the system, and are constantly being used to interpret new data. The power of ML algorithms mainly stems from the "learning rules" used for updating the weights. These rules are usually *local*, in the sense that they depend only on information available at the site of the *synapse*. Such local learning rules go under the general name of *Hebbian* learning rules, named after the physiologist Donald Hebb, who first suggested the distributed mechanism underlying neural function [12]. Hebbian computation was first implemented successfully by Rosenblatt in the canonical Perceptron algorithm [13], which spawned many other useful algorithms [14].

Implementing ML algorithms such as the Perceptron on conventional general-purpose digital hardware (*e.g.*, von Neumann architecture) is highly inefficient. A prime reason for this is the physical separation between the memory arrays used to store the values of the synaptic weights and the arithmetic module used to compute the update rules. General-purpose architecture actually eliminates the advantage of Hebbian learning rules - their locality, which allows highly efficient parallel computation. To overcome the inefficiency of general-purpose hardware, numerous dedicated hardware designs, based on CMOS technology, have been proposed in the past two decades. These designs perform online learning tasks, based on algorithms from ML [15, and references therein]. These designs use massively parallel *synaptic arrays*, where each synapse stores a synaptic weight and updates it locally. However, so far, these designs are not commonly used for practical large scale ML applications, and it is not clear whether they could be scaled up, since each synapse requires to much power and area (see section VII). This issue of scalability possibly casts doubt on the entire field [16].

Recently, it has been suggested [17] that scalable hardware implementations of ML algorithms may become possible if a novel device, the *memristor* [18–20], is used. A memristor is a resistor with a varying, history-dependent resistance. It is a passive analog device with activation-dependent dynamics, which makes it ideal for registering and updating of synaptic weights. Furthermore, its relatively small size enables integration of memory with the computing circuit [21] and allows a compact and efficient architecture for learning algorithms. To date, no circuit has been suggested to utilize memristors for implement-

D. Soudry, D. Di Castro, A. Gal, A. Kolodny and S. Kvatinsky are with the Department of Electrical Engineering, Technion - Israel institute of Technology, Haifa 32000, Israel (email: daniel.soudry@gmail.com)

ing general ML algorithms. Several circuits have been suggested for specific Spike Time Dependent Plasticity (STDP) like learning rules [22–25], which require "spiking neurons" and have a limited use from ML perspective [26]. This issue is discussed in detail in section VII.

The main challenge for general ML algorithm circuit design arises from the nature of Hebbian rules: practically all of them contain a multiplicative term [27], which is hard to implement in compact and scalable hardware. In this paper, a novel and general scheme to design hardware for Hebbian learning rules is presented. The proposed scheme uses a memristor as a memory element to store the weight and temporal encoding as a mechanism to perform a multiplication operation. The proposed design uses a single memristor and two CMOS transistors per synapse, and requires therefore 2% to 8% of the area and power of previously proposed CMOS-only circuits. The functionality of a circuit utilizing the memristive synapse array is demonstrated numerically by a handwritten digits recognition task, where the circuit performs as well as the software algorithm. Introducing noise levels of about 10% and parameter variability of about 30% did not affect significantly the performance of the circuit, due to the inherent robustness of ML algorithms. The proposed design may therefore allow the use of specialized hardware for ML algorithms, rather than the currently used general-purpose architecture.

The remainder of the paper is organized as follows: In section II basic background on memristors and ML algorithms is given; in section III the proposed circuit is explained; in section IV the circuit is evaluated numerically using a specific algorithm; in section V it is described how to implement general ML algorithms using the circuit; in section VI a few variations from the basic synaptic circuit are described; in section VII the novelty of the proposed circuit is discussed and compared with previous works; and in section VIII the paper is summarized.

II. BACKGROUND

For convenience, basic background information on memristors and ML algorithms is given in this section.

A. The memristor

A memristor [18, 19], originally proposed by Chua in 1971 as "the missing fourth fundamental element", is a passive electrical element. Memristors are basically resistors with varying resistance, where their resistance changes according to time integral of the current through the device, or alternatively, the integrated voltage upon the device. In the "classical" representation the conductance of a memristor G depends directly on the integral over time of the voltage upon the device, sometimes referred to as "Flux". Formally, a memristor obeys the following equations

$$i\left(t\right) \;=\; G\left(s\left(t\right)\right)v\left(t\right), \tag{1}$$

$$\dot{s}\left(t\right) \;=\; v\left(t\right). \tag{2}$$

A generalization of this model , which is called a *memristive system* [28], was proposed in 1976 by Chua and Kang. In memristive devices, s is a general state variable, rather than an integral of the voltage. Memristive devices are discussed in section VI. In the following sections it is assumed that the variations in the value of $s\left(t\right)$ are restricted to be small, so that $G\left(s\left(t\right)\right)$ can be linearized around some point s^{*} and the conductivity of the memristor is given, to first order, by

$$G\left(s\left(t\right)\right) = \bar{g} + \hat{g}\cdot s\left(t\right), \tag{3}$$

where $\hat{g} = \left[dG\left(s\right)/ds\right]_{s=s^{*}}$ and $\bar{g} = G\left(s^{*}\right) - \hat{g}\cdot s^{*}$.

B. Machine Learning

Machine Learning (ML), a branch of artificial intelligence, is dedicated to the construction and study of systems that can learn from data. For example, consider the following ML "supervised learning" task. Assume a system that operates on K discrete presentations of inputs ("trials"), indexed by $k = 1, 2, \ldots, K$. For brevity, the indexing of iteration number is sometimes suppressed, where it is clear from the context. On each trial k, the system receives empirical data, a pair of two real column vectors of sizes M and N: a *pattern* $\mathbf{x}^{(k)} \in \mathbb{R}^{M}$ and a *desired* label $\mathbf{d}^{(k)} \in \mathbb{R}^{N}$, with all pairs sharing the same desired relation $\mathbf{d}^{(k)} = f\left(\mathbf{x}^{(k)}\right)$. The objective of the system is to estimate ("learn") the function $f\left(\cdot\right)$ using the empirical data.

As a simple example, suppose $\mathbf{W}$ is a tunable $N \times M$ matrix of parameters, and consider the linear estimator

$$\mathbf{r}^{(k)} = \mathbf{W}^{(k)}\mathbf{x}^{(k)}, \tag{4}$$

or

$$r_{n}^{(k)} = \sum_{m} W_{nm}^{(k)} x_{m}^{(k)}\,. \tag{5}$$

The *result* of the estimator, $\mathbf{r} = \mathbf{W}\mathbf{x}$, should aim to predict the right *desired* labels $\mathbf{d} = f\left(\mathbf{x}\right)$ for new unseen *patterns* $\mathbf{x}$. To solve this problem, $\mathbf{W}$ is tuned to minimize some measure of error between the the estimated and desired labels, over a K_0-long subset of the empirical data, called the "training set" (for which

$k = 1, ..., K_0$). For example, if we define the *error* vector

$$\mathbf{y}^{(k)} \triangleq \mathbf{d}^{(k)} - \mathbf{r}^{(k)}, \tag{6}$$

then a common measure is the Mean Square Error (MSE)

$$\text{MSE} \triangleq \sum_{k=1}^{K_0} \left\| \mathbf{y}^{(k)} \right\|^2. \tag{7}$$

The performance of the resulting estimator is then tested over a different subset, called the "test set" ($k = K_0 + 1, ..., K$).

A reasonable iterative algorithm for minimizing objective (7) (*i.e.*, updating $\mathbf{W}$, where initially $\mathbf{W}$ is arbitrarily chosen) is the following *online gradient descent* (also called *stochastic gradient descent*) iteration

$$\mathbf{W}^{(k)} = \mathbf{W}^{(k-1)} - \frac{1}{2}\eta \nabla_{\mathbf{W}} \left\| \mathbf{y}^{(k)} \right\|^2, \tag{8}$$

where the $1/2$ coefficient is written for mathematical convenience, η is the *learning rate*, a (usually small) positive constant, and at each iteration k a single empirical sample is chosen randomly and presented at the input of the system. The gradient can be calculated by differentiation using the chain rule, (6) and (4). Defining $\Delta\mathbf{W}^{(k)} \triangleq \mathbf{W}^{(k)} - \mathbf{W}^{(k-1)}$, and $(\cdot)^\top$ to be the transpose operation, we obtain the *outer product*

$$\Delta\mathbf{W}^{(k)} = \eta \mathbf{y}^{(k)} \left(\mathbf{x}^{(k)} \right)^\top, \tag{9}$$

or

$$W_{nm}^{(k)} = W_{nm}^{(k-1)} + \eta x_m^{(k)} y_n^{(k)}. \tag{10}$$

Specifically, this update rule is called the "Adaline" algorithm [29], used in adaptive signal processing and control [30]. The parameters of more complicated (non-linear) estimators can also be similarly tuned ("trained"), using online gradient descent or similar methods (see section V). Online gradient descent is considered to be very effective in large scale problems [31], there are guarantees that this optimization procedure converges to some local minimum, and it is generally considered the best algorithm for training artificial neural networks [32]. Importantly, note that the update rule in (1) is "local"- *i.e.*, the change in the synaptic weight $W_{nm}^{(k)}$ depends only on the related components of input $\left(x_m^{(k)}\right)$ and error $\left(y_n^{(k)}\right)$. This local update, which appears in many other ML algorithms, enables a massively parallel hardware design, as explained in section III.

Such massively parallel designs are needed, since for large N and M, ML algorithms implementing supervised learning task may be computationally prohibitive in both time and memory space. For example, in the simple Adaline algorithm the main computational burden, in each iteration, comes from (5) and (10), where the number of operations (addition and multiplication) is of order $O(M \cdot N)$. Commonly, these steps have become the main computational bottleneck in executing ML algorithms in software (*e.g.*, the Backpropgation algorithm), as we show in section V. Other algorithmic steps, such as (6) here, are usually linear in either M or N, and therefore have a negligible computational complexity.

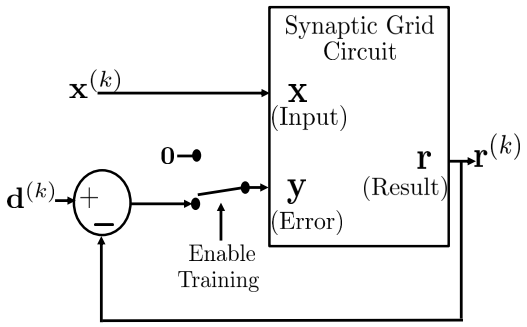

Fig. 1 A simple Adaline learning task, with the proposed "Synaptic Grid" circuit executing (4) and (9), which are the main computational bottlenecks in the algorithm.

III. Circuit design

In this section, dedicated analog hardware for implementing ML algorithms is proposed. A grid of "artificial synapses" is constructed, where each synapse stores to a single *synaptic weight* W_{nm}. The grid is a large $N \times M$ array of synapses, where the synapses operate simultaneously, each performing a simple local operation. This *synaptic grid* circuit carries the main computational load in ML algorithms by implementing the two computational bottlenecks, (5) and (10), in a massively parallel way. This matrix$\times$vector product in (5) is done using a resistive grid (of memristors), implementing multiplication through Ohm's law and addition through current summation. The vector$\times$vector outer product in (10) is done by using the fact that, given a voltage pulse, the conductivity of a memristor will increment proportionally to the pulse duration multiplied by the pulse magnitude. Using this method, multiplication requires only two transistors per synapse. Thus, together with a negligible amount of $O(M + N)$ additional operations, these arrays can be used to execute ML algorithms efficiently.

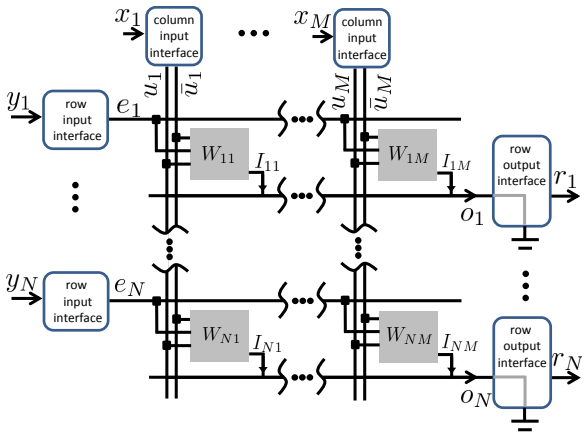

Fig. 2 Synaptic grid ($N \times M$) circuit architecture scheme. Every (n, m) node in the grid is a memristor-based synapse that receives voltage input from the shared $u_m, \bar{u}_m$ and the e_n lines and outputs I_{nm} current on the o_n lines. These output lines receive total current arriving from all the synapses on the n-th row, and are grounded.

A. Circuit function

Similarly to the Adaline algorithm described in section II, the circuit operates on discrete presentations of inputs ("trials") - see Fig. 1. On each trial k, the circuit receives an *input* vector $\mathbf{x}^{(k)} \in [-A,\ A]^M$ and an *error* vector $\mathbf{y}^{(k)} \in [-A,\ A]^N$ (where A is a bound on both the input and error) and produces a *result* output vector $\mathbf{r}^{(k)} \in \mathbb{R}^N$, which depends on the input by the relation (4), where the matrix $\mathbf{W}^{(k)} \in \mathbb{R}^{N\times M}$, called the *synaptic weight matrix*, is stored in the system. Additionally, on each step, the circuit updates $\mathbf{W}^{(k)}$ according to (9). This circuit can be used to implement ML algorithms. For example, as depicted in Fig. 1 the simple Adaline algorithm can be implemented using the circuit, with training enabled on $k = 1, \ldots, K_0$. The implementation of more complicated ML algorithms using this circuit, such as the Backpropgation algorithm, is shown in section V.

B. The circuit architecture

B.1 The synaptic grid

The system described in Fig. 1 is implemented by the circuit shown in 2, where components of all vectors are shown as individual signals. Each gray cell in Fig. 2 is a synaptic circuit ("artificial synapse") using a memristor (described in Fig. 3a). The synapses are arranged in a two dimensional $N \times M$ grid array as shown in Fig. 2, where each synapse is indexed by (n, m), with $m \in \{1..M\}$ and $n \in \{1..N\}$. Each (n, m) synapse receives two *inputs* $u_m, \overline{u}_m$, an *enable* signal e_n, and *output* current I_{nm}. Each column of synapses in the array (the m-th column) shares two vertical input lines u_m and $\overline{u}_m$, both connected to a "column input interface". The voltage signals u_m and $\overline{u}_m$ $(\forall m)$ are generated by the column input interfaces from the components of the input signal $\mathbf{x}$, upon presentation. Each row of synapses in the array (the n-th row) shares the horizontal enable line e_n and output line o_n, where e_n is connected to a "row input interface" and o_n is connected to a "row output interface". The voltage (pulse) signal on the enable line e_n $(\forall n)$ is generated by the row input interfaces from the error signal $\mathbf{y}$, upon presentation. The row output interfaces keep the $\mathbf{o}$ lines grounded ($V = 0$) and convert the total current from all the synapses in the row going to the ground, $\sum_m I_{nm}$, into the output signal $\mathbf{r}$.

B.2 The artificial synapse

The proposed memristive synapse is composed of a single memristor, connected to a shared terminal of two MOSFET transistors (P-type and N-type), as shown schematically in in Fig. 3a (without the n, m indices). These terminals act as drain or source, interchangeably, depending on the input, similarly to the CMOS transistors in transmission gates. Recall the memristor dynamics are given by (1-3), with $s(t)$ being the state variable of the memristor and $G(s(t)) = \bar{g} + \hat{g}s(t)$ its conductivity. Also, the current of the N-type transistor in the linear region is, ideally,

$$I = K\left(\left(V_{GS} - V_T\right)V_{DS} - \frac{1}{2}V_{DS}^2\right), \qquad (11)$$

where V_{GS} is the gate-source voltage, V_T is the voltage threshold, V_{DS} is the drain-source voltage, and K is the conduction parameter of the transistors. The current equation for the P-type transistor is symmetrical to (11), and for simplicity we assume that K and $|V_T|$ are equal for both transistors. The synapse receives three voltage input signals: u and $\bar{u} = -u$ are connected, respectively, to a terminal of the N-type and P-type transistors and an *enable* signal e is connected to the gate of both transistors. The enable signal can have a value of 0, V_{DD}, or $-V_{DD}$ (with $V_{DD} > |V_T|$) and have a pulse shape of varying duration, as explained below. The output of the synapse is a current I to the grounded line o. The magnitude of the input signal $u(t)$ and the circuit parameters are set so they fulfill the following assumptions

1. If $e = 0$ (*i.e.*, the gate is grounded) both transistors are non-conducting (in the cut-off region), *i.e.*,
$$|u(t)| < |V_T| \qquad (12)$$

2. If $e = \pm V_{DD}$, then, when in the linear region, both transistors have relatively high conductivity as compared to the conductivity of the memristor, *i.e.*,

$$K\left(V_{DD} - |u\left(t\right)| - |V_T|\right) \gg G\left(s\left(t\right)\right) \quad (13)$$

To satisfy (12) and (13), the proper value of $u(t)$ is chosen. Note (13) is a reasonable assumption, as shown in [33]. If not (*e.g.*, if the memristor conductivity is very low), instead one can use an alternative design, as described in appendix A [34]. Under these assumptions, when $e = 0$ then $I = 0$ in the output and the voltage across the memristor is zero. In this case, the state variable does not change. If $e = \pm V_{DD}$, from (13), the voltage on the memristor is approximately $\pm u$. When $e = V_{DD}$ the N-type transistor is conducting in the linear region while the P-type transistor is cut off. When $e = -V_{DD}$ the P-type transistor is conducting in the linear region while the N-type transistor is cut off.

C. Circuit operation

The operation of the circuit in each trial (a single presentation of a specific input) is composed of two phases. First, in the computing phase ("read"), the output current from all the synapses is summed and adjusted to produce an arithmetic operation $\mathbf{r} = \mathbf{W}\mathbf{x}$ from (4). Second, in the updating phase ("write"), the synaptic weights are incremented according to the update rule $\Delta\mathbf{W} = \eta\mathbf{x}\mathbf{y}$ from (10). In the proposed design, for each synapse, the synaptic weight, W_{nm}, is stored using s_{nm}, the memristor state variable of the (n, m) synapse. The parallel "read" and "write" operations are achieved by applying simultaneous voltage signals on the inputs u_m and enable signals e_n $(\forall n, m)$. The signals and their effect on the state variable are shown in Fig. 3b.

C.1 Computation phase ("read")

During each read phase, a vector $\mathbf{x}$ is given, and encoded in $\mathbf{u}$ and $\bar{\mathbf{u}}$ component-wise by the column input interfaces, for a duration of $T_{\rm rd}$, $\forall m: u_m\left(t\right) = ax_m = -\bar{u}_m\left(t\right)$, where a is a positive constant converting x_m, a unit-less number, to voltage. Recall that A is the maximal value of $|x_m|$, so we require that $aA < |V_T|$, as required in (12). Additionally, the row input interfaces produce voltage signal on the e_n lines, $\forall n$:

$$e_n\left(t\right) = \begin{cases} V_{DD} & , \text{ if } 0 \le t < 0.5T_{\rm rd} \\ -V_{DD} & , \text{ if } 0.5T_{\rm rd} \le t \le T_{\rm rd} \end{cases}. \quad (14)$$

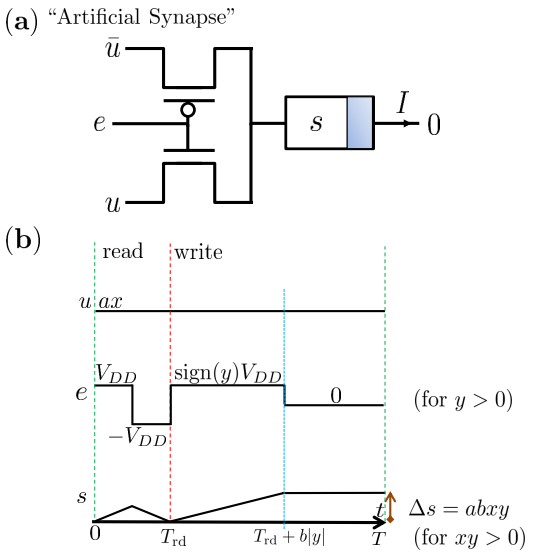

Fig. 3 Memristor-based synapse. (a) A schematic of a single memristive synapse (without the n, m indices). The synapse receives input voltages u and $\bar{u} = -u$, an enable signal e, and output current I. (b) The "read" and "write" protocols - incoming signals in a single synapse and the increments in the synaptic weight s as determined by (24). $T = T_{\rm wr} + T_{\rm rd}$.

From 2, the total change in the internal state variable is therefore, $\forall n, m$:

$$\Delta s_{nm} = \int_0^{0.5T_{\rm rd}} \left(ax_m\right) dt + \int_{0.5T_{\rm rd}}^{T_{\rm rd}} \left(-ax_m\right) dt = 0\,, \quad (15)$$

The zero net change in the value of s_{nm} between the times of 0 and $T_{\rm rd}$ implements a non-destructive "read". To minimize inaccuracies due to changes in the conductance of the memristor during the "read" phase, the row output interface samples the output current at time zero. The output current of the synapse to the o_n line at the time is thus

$$I_{nm} = a(\bar{g} + \hat{g}s_{nm})x_m. \quad (16)$$

Therefore, the total current in each output line o_n equals to the sum the individual currents produced by the synapses driving that line, *i.e.*,

$$o_n \;=\; \sum_m I_{nm} = a\sum_m \left(\bar{g} + \hat{g}s_{nm}\right) x_m\,. \quad (17)$$

The row output interface measures the output current o_n, and outputs

$$r_n = c\left(o_n - o_{\rm ref}\right) \quad (18)$$

where c is a constant converting the current units of o_n to a unit-less number r_n, and

$$o_{\mathrm{ref}} = a\bar{g}\sum_m x_m\,. \tag{19}$$

Defining

$$W_{nm} = ac\hat{g}s_{nm}\,, \tag{20}$$

we obtain

$$\mathbf{r} = \mathbf{W}\mathbf{x}\,, \tag{21}$$

as desired.

C.2 Update phase ("write")

During each write phase, of duration of T_{wr}, $\mathbf{u}, \bar{\mathbf{u}}$ maintain their values from the "read" phase, while the signal $\mathbf{e}$ changes. In this phase the row input interfaces encode $\mathbf{e}$ component-wise, $\forall n$: w

$$e_n(t) = \begin{cases} \mathrm{sign}(y_n)\, V_{DD} & ,\ \text{if } 0 \le t - T_{\mathrm{rd}} \le b|y_n| \\ 0 & ,\ \text{if } b|y_n| < t - T_{\mathrm{rd}} < T_{\mathrm{wr}} \end{cases} \tag{22}$$

The interpretation of (22) is that e_n is a pulse with magnitude V_{DD}, the same sign as y_n, and a duration $b|y_n|$ (where b is a constant converting y_n, a unit-less number, to time units). Recall that A is the maximal value of $|y_n|$, so we require that $T_{\mathrm{wr}} > bA$. The total change in the internal state is therefore

$$\Delta s_{nm} = \int_{T_{\mathrm{rd}}}^{T_{\mathrm{rd}}+b|y_n|} (a\,\mathrm{sign}(y_n)\, x_m)\, dt \tag{23}$$

$$= abx_m y_n \tag{24}$$

Using (20), the desired update rule for the synaptic weights is therefore obtained

$$\Delta\mathbf{W} = \eta\mathbf{y}\mathbf{x}^\top\,, \tag{25}$$

where $\eta = a^2bc\hat{g}$.

IV. Circuit Evaluation

In this section, the proposed circuit is implemented (section A), an online gradient descent learning algorithm (Adaline) is used to demonstrated the implementation of ML algorithms with the proposed circuit (section B), and the robustness of the circuit to noise and variation is evaluated (section C).

A. Circuit implementation

The proposed circuit has been implemented using Matlab in Simulink environment [35] using the SimElectronics toolbox [36] which includes standard MOSFET devices. The Matlab code generates a Simulink implementation of the proposed circuit for variable size (M and N), using the equations described in section III. As depicted in Fig. 2, the implementation consists of a synapse-grid and the interface blocks. The interface units were implemented using standard Simulink components. For example, the input and output row interface units where implemented together: the pulse width modulation was implemented using a thresholded sawtooth control signal, and the voltage values of e_n and the current values of o_n are both are multiplied by $\mathrm{sign}(y_n)$, during the "read" phase. Also, the input and output to the circuit were kept constant in each trial using "sample and hold" units.

The memristor model is implemented using (1-3) with $s(0) = 0$, and parameters $\bar{g} = 10^{-6}\,\Omega^{-1}$ and $\hat{g} = 18\cdot 10^{-5}\,[\Omega\cdot\mathrm{V}\cdot\mathrm{sec}]^{-1}$ taken roughly from experimental data ([37], Fig. 2). The parameters of the (ideal) transistors are kept at their defaults: $K = 5\,\mathrm{AV}^{-2}$ and $V_T = 1.7\,\mathrm{V}$ for the N-type transistor, $K = 5\,\mathrm{AV}^{-2}$ and $V_T = -1.4\,\mathrm{V}$ for the P-type transistor. The rest of the circuit parameters are set as $V_{DD} = 10\mathrm{V}, a = 1\mathrm{mV}, b = T_{\mathrm{wr}}, c = 100A^{-1}, T_{\mathrm{wr}} = 0.6T$ and $T_{\mathrm{rd}} = 0.2T$ with $T = 0.1\,\mathrm{sec}$ being the duration of each trial (a period of $0.2T$ was used for signal sampling in the "sample and hold" units).

These specific parameters, however, are not strictly necessary for proper execution of the proposed design, and are only used to demonstrate its applicability. In fact, it is straightforward to show that K, V_T and $\bar{g}$ have little effect (as long as (12-13) hold), and different values of $\hat{g}$ can be adjusted for by rescaling c. This is important since the feasible range of parameters for the memristive devices is still not well characterized, and it seems to be quite broad. For example, the values of the memristive timescales range from picoseconds [38] to milliseconds [37]. The duration of each trial T should approximately decrease together with the timescale of the memristor (here we used parameters from millisecond-timescale memristor [37]). The circuit operation is demonstrated in Fig. 4, where a 2×2 synaptic grid circuit is simulated for time $10T$ with simple inputs $x_1 = 10\mathrm{sign}\,(\mathrm{t} - 5\mathrm{T})$,$x_2 = -20\mathrm{sign}\,(\mathrm{t} - 5\mathrm{T})$,$y_1 = 0.5$, and $y_2 = -0.25$.

B. Evaluating circuit performance

To evaluate circuit performance, a standard handwritten digits recognition task is used. The aim of this task is to train an artificial neural network to correctly identify handwritten digits based on labeled $(0, 1, \ldots, 9)$ examples from the MNIST database, (a classical ML test [39], available at [40]) of 28×28

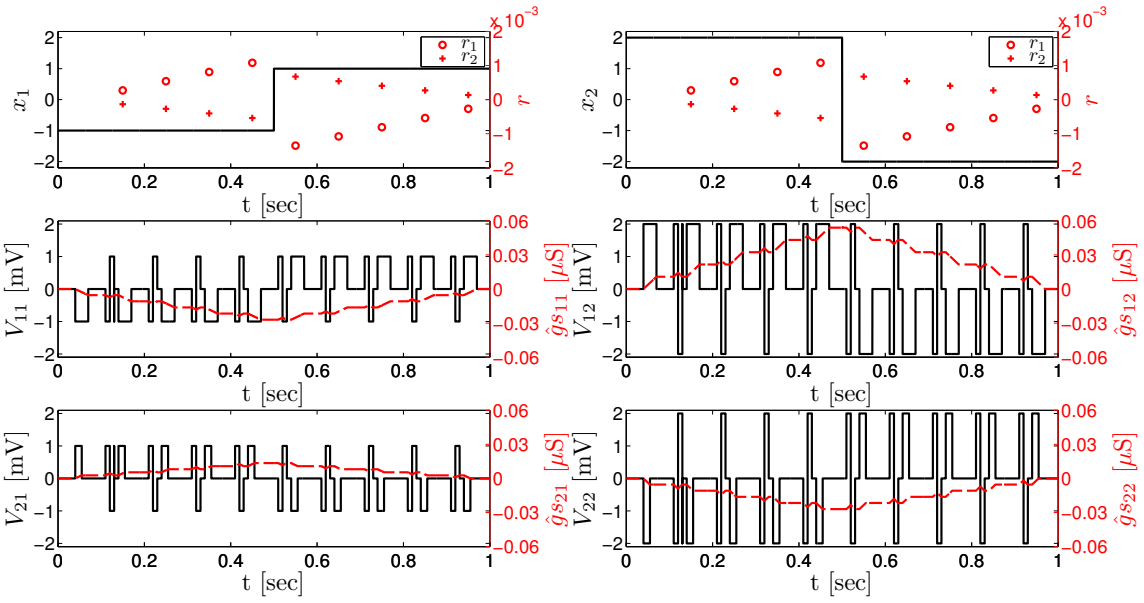

Fig. 4 Synaptic 2×2 grid circuit simulation, during ten operation cycles. *Top:* Circuit Inputs (x_1, x_2) and result outputs (r_1, r_2). *Middle and bottom:* voltage (solid black) and conductance (dashed red) change for each memristor in the grid (error input: $(y_1, y_2) = (0.5, -0.25)$).

grayscale pictures, (Fig. 5a). Each picture is converted to a 784-long vector $\mathbf{x}$, and each label is converted to a 10-long vector $\mathbf{d}$. Following the standard procedure in neural networks training [32], the pictures are uncorrelated using principal component analysis (which is, incidentally, another algorithm that can be implemented by Hebbian learning [41]). To train the network, 9000 labeled examples (900 for each label) are presented repeatedly (six times) at random order. To test the network a different set of 1000 labeled examples is presented (100 for each label).

The task is to find, based on the training data, a weight matrix $\mathbf{W}$ so that, for any picture from the test set, the predictor $\mathbf{r}^{(k)} = \mathbf{W}\mathbf{x}^{(k)}$ will approximate $\mathbf{d}^{(k)}$ and allow to estimate the correct label with minimal probability of error. In this task, if $\mathbf{W} \in \mathbb{R}^{10\times 785}$, *i.e.*, it is fully connected to all components of the input $\mathbf{x}$, the classifier (a single layer neural network) can achieve a minimal error of 12% [39]. To reduce simulation run time of the circuit, a smaller version of the basic, fully connected, single layer classifier is implemented. Instead of using all of the 784 components of $\mathbf{x}$, only the first 29 principal components are used as input, together with a constant input of value 1 ("bias"). This gives a 30×10 network with 300 free parameters, instead of 7850 free parameters in the original fully connected layer.

The network has been trained using the Adaline algorithm (see section II, and Fig. 1). At each trial k: (1) $\mathbf{x}^{(k)}$ is given as input to the circuit, (2) the result $\mathbf{r}^{(k)} = \mathbf{W}^{(k)}\mathbf{x}^{(k)}$ is read from the output of the circuit, (3) $\mathbf{y}^{(k)} = \mathbf{t}^{(k)} - \mathbf{r}^{(k)}$ is given as input to the circuit, and (4) the weight matrix is incremented according to $\Delta\mathbf{W}^{(k)} = \eta\mathbf{y}^{(k)}\left(\mathbf{x}^{(k)}\right)^{\top}$ with $\eta = 31.5$. The proposed circuit has been compared to a direct implementation of the software algorithm using Matlab. The performances of the circuit and the software algorithm improve similarly during the training phase, as shown in Fig. 5b. The performance is maintained in the test phase, *i.e.*, when the training is stopped, $\mathbf{W}$ is no longer updated and new (unseen) examples are presented. Both perform with approximately 17% chance of error, sufficiently close to 12%, the minimum achievable error of the fully connected (approximately 26 times larger) single layer classifier.

C. Evaluating robustness to noise and variability

Usually, analog computation suffers from reduced robustness to noise as compared to digital computation [42]. ML algorithms are, however, inherently robust to noise, which is a key element in the set of problems they are designed to solve (*e.g.*, the hand written digits in the example above are noisy). This suggests that the effects of intrinsic noise on the performance of the analog circuit are relatively small. These effects largely depend on the specific circuit implemen-

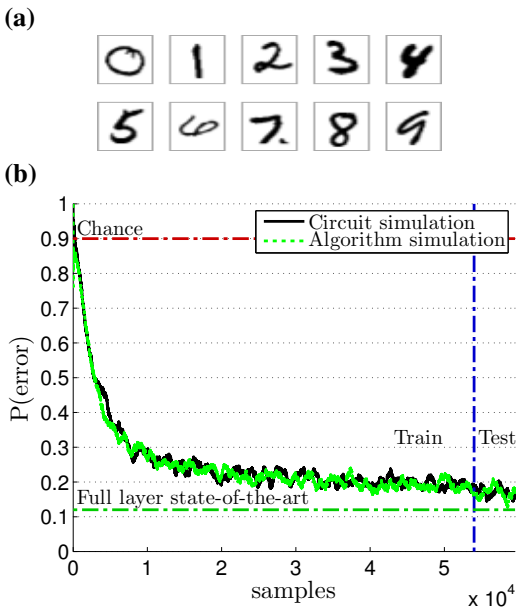

Fig. 5 Handwritten digits recognition task. (a) A sample of handwritten digits used to test the circuit. The pictures and labels are respectively converted into $\mathbf{x}^{(k)}$ and $\mathbf{d}^{(k)}$, which are used to train the proposed circuit as in Fig. 1, with $M = 30$ and $N = 10$. (b) Performance of circuit simulation is similar to the same algorithm run directly in software. For comparison, error probability is given for random classification ("chance"), and full single layer ($M = 785, N = 10$). Training stops at vertical line.

tation (*e.g.*, the CMOS process). Particularly, memristor technology is not mature yet and memristors have not been fully characterized. To check the robustness of the circuit, crude estimation of the magnitude of noise and variability has been used. This estimation is based on known sources of noise and variability, which are less dependent on specific implementation. The alleged robustness of the circuit is evaluated by simulating the circuit in the presence of these noise and variation sources.

C.1 Noise

When one of the transistors is enabled ($e(t) = \pm V_{DD}$), then the current is affected by intrinsic thermal noise sources in the transistors and memristor of each synapse. Current fluctuations on a device due to thermal origin can be approximated by a white noise signal $I(t)$ with zero mean ($\langle I(t)\rangle = 0$) and autocorrelation $\langle I(t) I(t')\rangle = \sigma^2 \delta(t - t')$ where $\delta(\cdot)$ is Dirac's delta function and $\sigma^2 = 2k\tilde{T}g$ (where k is Boltzmann constant, $\tilde{T}$ is the temperature, and g is the conductance of the device). For $65\,\mathrm{nm}$ transistors (parameters taken from IBM's 10LPe/10RFe process [43]) the characteristic conductivity is $g_1 \sim 10^{-4}\,\Omega^{-1}$, and therefore for $I_1(t)$, the thermal current source of the transistors are $\sigma_1^2 \sim 10^{-24}\mathrm{A}^2\mathrm{sec}$. Assume that the memristor characteristic conductivity $g_2 = \epsilon g_1$, so for the thermal current source of the memristor, $\sigma_2^2 = \epsilon\sigma_1^2$. Note that from (13) we have $\epsilon \ll 1$, the resistance of the transistor is much smaller than that of the memristor. The total voltage on the memristor is thus

$$\begin{aligned} V_M(t) &= \frac{g_1}{g_1+g_2}u(t) + (g_1+g_2)^{-1}(I_1(t) - I_2(t)) \\ &= \frac{1}{1+\epsilon}u(t) + \xi(t) \end{aligned}$$

where $\xi(t) = g_1^{-1}(I_1(t) - I_2(t))$ and $\langle \xi(t)\xi(t')\rangle \sim \sigma_\xi^2\delta(t-t')$ with $\sigma_\xi^2 \approx 2k\tilde{T}g_1^{-1} \sim 10^{-16}\mathrm{V}^2\mathrm{sec}$. The equivalent circuit, including sources of noise, is shown in Fig. (6). Assuming the circuit minimal trial duration is $T = 10\mathrm{nsec}$, the maximum root mean square error due to thermal noise would be about $E_T \sim T^{-1/2}\sigma_\xi \sim 10^{-4}V$.

Noise in the inputs $u, \bar{u}$ and e also exists. According to [44], the relative noise in the power supply of the $u/\bar{u}$ inputs is approximately 10% in the worst case. Applying $u = ax$, effectively gives an input of $ax + E_T + E_u$, where $|E_u| \le 0.1a|x|$. The absolute noise level in duration of e should be smaller than $T_{\mathrm{clk}}^{\min} \sim 2\cdot 10^{-10}\mathrm{sec}$, assuming a digital implementation of pulse-width modulation with $T_{\mathrm{clk}}^{\min}$ being the shortest clock cycle currently available. On every write cycle $e = \pm V_{DD}$ is therefore applied for a duration of $b|y| + E_e$ (instead of $b|y|$), where $|E_e| < T_{\mathrm{clk}}$.

When running the learning algorithm in software, these noise sources can be simulated by similar changes in the input. Based on both estimations, the algorithm and the circuit, with additional noise sources, according to the analysis above, has been executed and evaluated. The performance of the software algorithm or the circuit has is identical to the performance without noise, as shown in Fig. 5b.

C.2 Parameter variability

A common estimation of the variability in memristor parameters is a Coefficient of Variation (CV = (standard deviation)/mean) of a few percent [45]. In this paper, the circuit is simulated with considerably larger variability ($CV \sim 30\%$), in addition to the noise sources as described in section C.1. This is done by assuming that the variability in the parameters of the memristors are random variables independently sampled from a uniform distribution $\hat{g} \sim U[0.5\hat{g}, 1.5\hat{g}]$. The rate of change of the memristors state is similarly

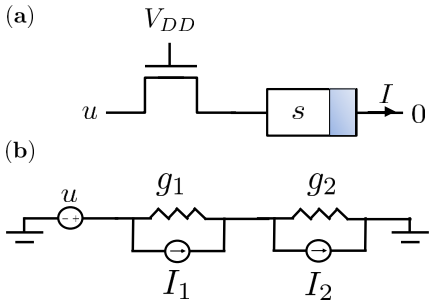

Fig. 6 Noise model for artificial synapse. (a) During operation, only one transistor is conducting (assume it is the N-type transistor). **(b)** Thermal noise in a small signal model, the transistor is converted to a resistor (g_1) in parallel to current source (I_1), the memristor is converted to a resistor (g_2) in parallel to current source (I_2), and the effects of the sources are summed linearly.

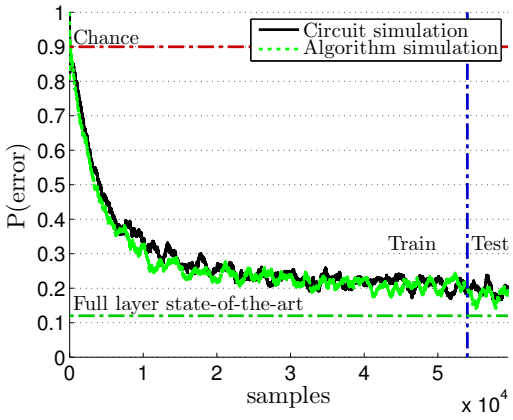

Fig. 7 The robustness of the ML algorithm - handwritten digits recognition task (Fig. **5**) with added $\sim$ 10% noise and $\sim$ 30% variability, resulting in a mild decrease in performance.

randomized, by rescaling the units of t in (2) ($\dot{s} = \alpha v$ with $\alpha = U[0.5, 1.5]$) independently in each memristor. When running the algorithm in software, these variations are equivalent to corresponding changes in the synaptic weights W and the learning rate η in the algorithm. As shown in Fig. 7b, in both cases, these variations had only a mild effect on the circuit performance. Note that variability in the transistor parameters is not considered, since these can affect the circuit operation only if (12) or (13) are invalidated. This can happen, however, only if the values of K or V_T vary in orders of magnitude, which is unlikely.

V. Implementing Machine Learning Algorithms

So far, the circuit operation was exemplified using the simple Adaline algorithm. In this section it is explained how, with a few adjustments, the proposed circuit can be used to implement various ML algorithms, besides Adaline. Recall the context of the "supervised learning" setup detailed in section II.

A. The Perceptron algorithm

The classical Perceptron algorithm [13], is not derived from online gradient descent (8). Its form is, however, very similar to the simple Adaline presented in the paper. In this algorithm the estimator function is $\mathbf{r} = \sigma(\mathbf{W}\mathbf{x})$ with a parameter matrix $\mathbf{W}$ and some sigmoidal function $\sigma(\cdot)$ that operates component-wise on the its vector input. Every iteration, the algorithm updates the output $\mathbf{r} = \sigma(\mathbf{W}\mathbf{x})$ and $\Delta\mathbf{W} = \eta\mathbf{y}\mathbf{x}^\top$, with $\mathbf{y} = \sigma(\mathbf{d} - \mathbf{r})$. The only additional operation, in comparison to the Adaline algorithm, is the $O(N)$ application of the $\sigma(\cdot)$ function on its $\mathbf{y}$ error vector input. This is negligible in comparison to $O(M \cdot N)$. Therefore, no modification is required to the circuit itself.

B. The Backpropagation algorithm

Consider a estimator of the form $\mathbf{r} = \mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x})$, with $\mathbf{W}_1$ and $\mathbf{W}_2$ being the two parameter matrices. This *double layer neural network* can approximate any target function with arbitrary precision [46]. Denote by $\mathbf{r}_1 = \mathbf{W}_1\mathbf{x}$ and $\mathbf{r}_2 = \mathbf{W}\sigma(\mathbf{r}_1)$ the output of each layer. In the Backpropagation algorithm, each update of the parameter matrices is given by an online gradient decent step, from (8),

$$\Delta\mathbf{W}_1 = \eta\mathbf{y}_1\mathbf{x}_1^\top \; ; \; \Delta\mathbf{W}_2 = \eta\mathbf{y}_2\mathbf{x}_2^\top \, ,$$

where $\mathbf{x}_2 = \sigma(\mathbf{r}_1)$, $\mathbf{y}_2 = \mathbf{d} - \mathbf{r}_2$, $\mathbf{x}_1 = \mathbf{x}$ and $\mathbf{y}_1 = \left(\mathbf{W}_2^\top\mathbf{y}_2\right) \odot \sigma'(\mathbf{r}_1)$, with $\mathbf{a} \odot \mathbf{b} \triangleq (a_1b_1, \ldots, a_Mb_M)$, a component-wise product. Implementing such an algorithm requires a minor modification of the proposed circuit - it should have an additional "inverted" output $\boldsymbol{\delta} \triangleq \mathbf{W}^\top\mathbf{y}$. Once this modification is made to the circuit, by cascading such circuits, it is straightforward to implement the Backpropagation algorithm for two layers or more, as shown in Fig. 8. Such "deep" multilayer neural networks are becoming increasingly useful [47], producing competitive solutions in many fields, such as pattern recognition [1, 3], natural language processing [4, 5] and predictions [6] (see [10] for related press). When the Backpropgation algorithm is executed with massive computational power it achieves state-of-the-art-results (*e.g.*, [9, 48]) using deep multilayer networks.

The additional output $\boldsymbol{\delta} = \mathbf{W}^\top\mathbf{y}$ can be generated by the circuit in an additional "read" phase with duration T_{rd}, between the original "read" and "write"

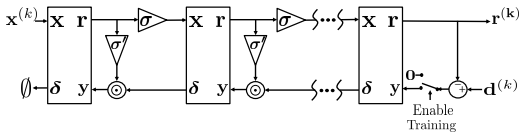

Fig. 8 Implementing the Backpropgation for a multilayer neural network, using a modified version of the original circuit. The triangles denote an operation of the function on the input (either $\sigma(\cdot)$ or $\sigma'(\cdot)$), and the $\odot$ circle denotes a component-wise product.

phases, in which the original role of the input and output lines are inverted. In this phase the NMOS transistor is on, *i.e.*, $\forall n$, $e_n = V_{DD}$, and the former "output" o_n lines are given the following voltage signal (used for a non-destructive read)

$$o_n = \begin{cases} ay_n & , \text{ if } T_{\rm rd} \leq t < 1.5T_{\rm rd} \\ -ay_n & , \text{ if } 1.5T_{\rm rd} \leq t \leq 2T_{\rm rd} \end{cases} .$$

The I_{nm} current now exits through the (original "input") u_m terminal, and the sum of all the currents is measured at time $T_{\rm rd}$ by the column interface (before it goes into ground)

$$u_m \ = \ \sum_n I_{nm} = a \sum_n \left(\bar{g} + \hat{g} s_{nm}\right) y_n \, . \quad (26)$$

The the total current on u_m at time $T_{\rm rd}$ is the output

$$\delta_m = c\left(u_m - u_{\rm ref}\right) \, , \quad (27)$$

where $u_{\rm ref} = a\bar{g} \sum_n y_n$. Thus, from 20,

$$\delta_m \ = \ \sum_n W_{nm} y_n \, ,$$

as required.

C. Other Algorithms

The use of the proposed circuit for additional algorithms can be easily extended if the input pattern $\mathbf{x}$ is replaced with $\phi(\mathbf{x})$, where ϕ is a "kernel function", operating component-wise. This simple extension covers many useful ML algorithms [49]. The proposed circuit may be used also for other algorithms which implement outer-products in their update rules, even outside of the "supervised learning" setting. For example, more advanced versions of the Perceptron or the Backpropgation algorithm, where the learning rate η is modified in an adaptive manner using a negligible amount of operations (*e.g.*, $O(M)$ in [50]). This is possible because η can become a tunable parameter in the proposed circuit by adjusting the tunable circuit parameters a, b or c. Additionally, "unsupervised learning" algorithms often include outer products, and may be implemented using the proposed circuit. For example, consider the the Oja rule [41] - an important online implementation of the principal components analysis.

VI. Circuit Modifications

In this section, a few additional important modifications of the proposed circuit are examined in more detail. In section A it is explained how to modify the circuit so it would work with more realistic "memristive devices" [20, 28], instead of the classical memristor model [18], given a few conditions. In section B, it is shown that it is possible to reduce the transistor count from two to one, at the price of doubling the duration of the "write" phase. Note that various other useful modifications of circuit are also possible. For example, the input $\mathbf{x}$ may be allowed to receive different values during the "read" and "write" operations. Also, it is straightforward to replace the simple outer product update rule in (10) by more general update rules of the form

$$W_{nm}^{(k)} = W_{nm}^{(k-1)} + \eta \sum_{i,j} f_i\left(y_n^{(k)}\right) g_j\left(x_m^{(k)}\right) \, ,$$

where f_i, g_j are some functions.

A. Generalization from "memristor" to "memristive devices"

In this paper, the memristor device is assumed to behave according to its classical model [18]. Though the first fabricated memristor device [19] has been modeled according to the classical model, this model is inaccurate and real devices can be modeled by the more general "memristive device" model. Furthermore, emerging memory technologies, *e.g.*, Resistive RAM and Spin-Torque Transfer MRAM, can be represented as memristive systems [20, 51]. A Memristive device [28] is a generalization of the original memristor [18]. For such devices the state variable can be a vector $\mathbf{s} \in \mathbb{R}^D$, and (assuming stationary dynamics)

$$\dot{\mathbf{s}} \ = \ \mathbf{f}\left(\mathbf{s}, u\right) \quad (28)$$

$$y \ = \ g\left(\mathbf{s}, u\right) u \, , \quad (29)$$

where u is the input (voltage/current) and y is the output (current/voltage). In [20], for example, s is a scalar as in the original memristor, but its dynamics affected by some "window" function $\Theta(s)$

$$\dot{s} = f\left(u\right) \cdot \Theta\left(s\right) \, . \quad (30)$$

Assume that $\Theta(s) > 0$ for all s and define $z(s) = \int_0^s (1/\Theta(x))\,dx$. Observe that

$$\dot{z} = (1/\Theta(s))\,\dot{s} = f(u)\,. \tag{31}$$

Additionally, since $z(s)$ is defined by an integral over a positive function, it is strictly monotone and therefore reversible to $s = h(z)$. Hence, it can be represented as

$$\dot{z} = f(u) \tag{32}$$
$$y = g(h(z), u)\,u\,. \tag{33}$$

Now this system is mathematically similar to the original system (1-2) . This allows to implement a Hebbian network in a similar method as for memristors, with z replacing s as the synaptic weight.

If u is the voltage and y current, A similar method as in the original memristor case is used. Assume a sufficiently small input range in which f is reversible, the following definitions are made, in order to have similar method as for the original memristor. During the write cycle replace the signals $u(t) = ax$ and $\bar{u}(t) = -ax$, respectively, with $u(t) = f^{-1}(ax)$and $\bar{u}(t) = f^{-1}(-ax)$. During the read cycle keep $u(t) = ax$ and replace the signal $\bar{u}(t) = -ax$ with $\bar{u}(t) = f^{-1}(-f(ax))$. Additionally, if for a sufficiently small range of state space , $g(h(z), u)$ is linearized and obtain $g(h(z), u) = \bar{g} + \hat{g}z + \gamma u$. The γu correction that does not appear in the memristor case, but may be corrected for by adjusting the reference signal o_{ref}. If u is current and y is voltage, a different design for the synapse should be used, as explained in appendix B [34].

B. Compact synapses

It is possible to reduce the number of transistors in each synapse from two to one. The schematic of such a synapse is shown in Fig. 9a. In this compact synapse the write time is double as compared to the original proposed synapse.. For simplicity, assume a classical memristor as in 1-2.

As depicted in Fig. 9b, the read cycle is performed by applying, for a T_{rd} duration,

$$u(t) = \begin{cases} ax & ,\ \text{if } 0 \le t < 0.5T_{\mathrm{rd}} \\ -ax & ,\ \text{if } 0.5T_{\mathrm{rd}} \le t \le T_{rd} \end{cases}, \tag{34}$$

and $e(t) = V_{DD}$, so $\dot{s}(t) = u(t)$ and $\Delta s = 0$ over the read cycle. Sampling the current at the beginning of the read cycle gives

$$I = a(\bar{g} + \hat{g}s)x = Wx\,, \tag{35}$$

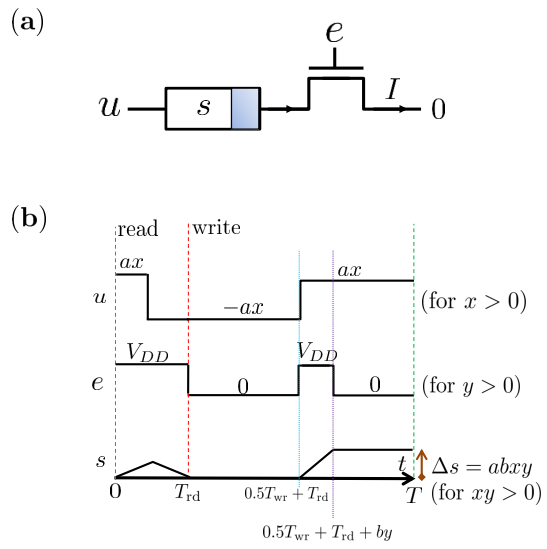

Fig. 9 Compact synapse design. (a) Schematic of the artificial synapse with input voltages u control signal e and output current I. **(b)** Writing and reading protocol - incoming signals in a single synapse and the increments in the synaptic weight s

as required.

In the write cycle

$$u(t) = \begin{cases} -ax & ,\ \text{if } T_{\mathrm{rd}} \le t \le T_{\mathrm{rd}} + 0.5T_{\mathrm{wr}} \\ ax & ,\ \text{if } T_{\mathrm{rd}} + 0.5T_{\mathrm{wr}} < t < T \end{cases}, \tag{36}$$

and $e(t) = V_{DD}$ if

$$\min(by, 0) \le t - T_{\mathrm{rd}} - 0.5T_{\mathrm{wr}} \le \max(by, 0) \tag{37}$$

and zero otherwise. Therefore, $\dot{s}(t) = ax$ if $e(t) = V_{DD}$, and zero otherwise.

Integrating over both the write and read cycles, we obtain again

$$\Delta s = abxy\,. \tag{38}$$

as required.

VII. Discussion

As explained in section II and V, two major computational bottlenecks of many Machine Learning (ML) algorithms are given by "matrix×vector" product operation (4) and a "vector×vector" outer product operation (9). Both are of order $O(M \cdot N)$, where M and N are the sizes of the input and output vectors. In this paper, the proposed circuit is designed specifically to deal with these bottlenecks using memristor arrays. This design has a relatively small number of components in each array element - one memristor

TABLE I Hardware designs of artificial synapses implementing scalable online learning algorithms

Design	#Transistors	Comments
Proposed design	2 (+1 memristor)	
[54]	2	Also requires UV light + Weights decay ~ minutes
[55]	6	Weights only increase (unusable)
[56, 57]	39	Must keep training
[58]	52	Must keep training
[59]	92	Weights decay ~ hours
[60]	83	Also requires a "weight unit"
[61]	150	

and two transistors. The physical grid-like structure of the arrays implements the "matrix$\times$vector" product operation (4) using analog summation of currents, while the memristor dynamics enable us to perform the "vector$\times$vector" outer product operation (10), using "time$\times$voltage" encoding paradigm.

The idea to use a resistive grid to perform "matrix$\times$vector" product operation is not new (*e.g.*, [52]). The main novelty of this paper is the use of memristors together with "time$\times$voltage" encoding, which allows us to perform a mathematically accurate "vector$\times$vector" outer product operation in the learning rule using a small number of components. Previous implementations of learning rules using memristor arrays [22–25] have been limited to spike-like inputs and focused on Synaptic Time Dependent Plasticity (STDP). Applications of STDP are usually aimed to explain biological neuroscience results. At this point, however, it is not clear how useful is STDP algorithmically. For example, the convergence of STDP-based learning is not guaranteed for general inputs (as it is guaranteed for the algorithms used in this work) [53]. Furthermore, to the best of the authors' knowledge, STDP-based algorithms are not yet competitive and have not been used yet for massive large scale problems - in contrast to the standard gradient-decent (Hebbian) learning rules discussed in this paper (see section V).

A. Previous CMOS-based designs

As mentioned in the introduction, CMOS hardware designs that specifically implement Hebbian learning algorithms remain an unfulfilled promise at this point.

The main incentive for existing hardware solutions is the inherent inefficiency in implementing these algorithms in software running on general purpose hardware (*e.g.*, CPU's, DSP's and GPU's). However, squeezing the required circuit for both the computation and the update phases (two configurable multipliers and a memory element) into an array cell has proven to be a hard task, using currently available CMOS technology. Off-chip or chip-in-the-loop design architectures (*e.g.*, [62], Table 1) have been suggested in many cases as a way around this design barrier. These designs, however, generally deal with the computational bottleneck of the "matrix$\times$vector" product operation in the computation phase, rather than the computational bottleneck of the "vector$\times$vector" outer product operation in the update phase. Additionally, these solutions are only useful in cases where the training is not continuous and is done in a pre-deployment phase or on special reconfiguration phases during the operation. Other designs implement non-standard (*e.g.*, perturbation algorithms [63]) or specifically tailored learning algorithms (*e.g.*, modified Backpropagation for spiking neurons [64]). However, it remains to be seen whether such algorithms are indeed scalable.

Hardware designs of artificial synaptic arrays that are capable of implementing common (scalable) learning algorithms (basically, online gradient-descent based learning, as explained in sections II and V) are listed in Table I. The effective transistor count per synapse (where resistors and capacitors were also counted as transistors) is approximately proportional to the required area and average power usage.

The smallest synaptic circuit [54], includes two transistors, similarly to our design, but requires the (rather unusual) use of UV illumination during its operation, and has the disadvantage of having volatile weights, decaying within minutes. The next device [55] includes six transistors per synapse, but the update rule can only increase the synaptic weights, which makes the device unusable for practical purposes. The next device [57, 65] suggested a grid design using CMOS Gilbert multipliers, resulting in 39 transistors per synapse. In a similar design, [58], 52 transistors are used. Both these devices use capacitive elements for analog memory and suffer from the limitation of having volatile weights, vanishing after training has stopped. They require therefore constant re-training (practically acting as "refresh"). Such retraining is required also on each start-up, or, alternatively, reading out the weights into an auxiliary memory - a solution which requires a mechanism for reading out the synaptic weights. The larger design in [59] (92 transistors) also has weight decay, but with a slow hours-long timescale. The device in [60] (83 transistors + an unspecified "weight unit" which stores the weights)

does not report to have weight decay, apparently since digital storage is used. This is also true for [61] (150 transistors).

The proposed memristor-based design should resolve these obstacles, and provide a compact, non-volatile circuit. Area and power consumption are expected to be reduced by a factor between 13 to 50, in comparison to standard CMOS technology, if a memristor is counted as an additional transistor (although it is actually more compact). Maybe the most convincing evidence for the limitations of these designs is the fact that, although most of these designs are two decades old, they have not been incorporated into commercial products. It is only fair to mention at this point, that while our design is purely theoretical and based on speculative technology, the above reviewed designs are based on mature technology, and have overcome obstacles all the way to manufacturing.

VIII. Conclusions

A novel method to implement scalable machine learning algorithms through Hebbian rules is proposed, based on the emerging memristor technology. The proposed method is based on an "artificial synapse" composed of one memristor, to store the "synaptic weight", and two CMOS transistors, to control the circuit. The circuit is estimated to be significantly smaller than existing CMOS-only designs, opening the opportunity for massive parallelism with millions of adaptive synapses on a single integrated circuit

The correctness of the proposed synapse structure exhibits similar accuracy to its equivalent software implementation, while the proposed structure shows extremely high robustness and immunity to noise and parameter variability. Building larger networks to execute similar algorithms can further increase accuracy to near-human performance[48].

Using the proposed memristive synapse design to execute machine learning algorithms, can give a significant boost to artificial intelligence with massive parallelism, high accuracy, low power, and good robustness.

References

[1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec. 1989.

[2] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. New York, NY, USA: John Wiley & Sons Inc, 2001.

[3] R. Socher and C. C. Lin, "Parsing natural scenes and natural language with recursive neural networks," *Proceedings of the 26th International Conference on Machine Learning (ICML)*, vol. 2, p. 7, 2011.

[4] R. Socher, C. Manning, and A. Y. Ng, "Learning continuous phrase representations and syntactic parsing with recursive neural networks," in *NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*, 2010, pp. 1–9.

[5] R. Socher and E. H. Huang, "Dynamic pooling and unfolding recursive autoencoders for paraphrase detection," *Advances in Neural Information Processing Systems*, vol. 24, pp. 801–809, 2011.

[6] D. A. Jiménez, "Fast path-based neural branch prediction," in *MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on. IEEE*, 2003, pp. 243–252.

[7] Q. V. Le, R. Monga, M. Devin, and G. Corrado, "Building high-level features using large scale unsupervised learning," in *ICML '12*, 2012, pp. 81–88.

[8] D. Ciresan, "Multi-column deep neural networks for image classification," in *CVPR '12*, 2012, pp. 3642–3649.

[9] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Ng, "Large scale distributed deep networks," *NIPS*, pp. 1–11, 2012.

[10] R. D. Hof, "Deep Learning," *MIT Technology Review*, 2013.

[11] D. Hernandez, "Now You Can Build Google's 1M Artificial Brain on the Cheap," *Wired*, 2013.

[12] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Mahwah, NJ, USA: Lawerence Erlbaum Associates, Inc., 2002.

[13] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[14] R. Rojas, *Neural networks: a systematic introduction*. Verlag Berlin Heidelberg: Springer, 1996.

[15] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomputing*, vol. 74, no. 1-3, pp. 239–255, Dec. 2010.

[16] A. R. Omondi, "Neurocomputers: a dead end?" *International journal of neural systems*, vol. 10, no. 6, pp. 475–481, 2000.

[17] M. Versace and B. Chandler, "The brain of a new machine," *Spectrum, IEEE*, vol. 47, no. 12, pp. 30–37, 2010.

[18] L. Chua, "Memristor-the missing circuit element," *Circuit Theory, IEEE Transactions on*, vol. 18, no. 5, pp. 507–519, 1971.

[19] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[20] 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, 2013.

[21] S. Kvatinsky, Y. H. Nacson, Y. Etsion, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based Multithreading," *Computer Architecture Letters*, vol. PP, no. 99, p. 1, 2013.

[22] D. Querlioz and O. Bichler, "Simulation of a memristor-based spiking neural network immune to device variations," *Neural Networks (IJCNN), The 2011 International Joint Conference on. IEEE*, pp. 1775–1781, 2011.

[23] C. Zamarreño Ramos, L. A. Camuñas Mesa, J. A. Pérez-Carrasco, T. Masquelier, T. Serrano-Gotarredona, and B. Linares-Barranco, "On spike-timing-dependent-plasticity, memristive devices, and building a self-learning visual cortex," *Frontiers in neuroscience*, vol. 5, p. 26, Jan. 2011.

[24] A. Nere, U. Olcese, D. Balduzzi, and G. Tononi, "A neuromorphic architecture for object recognition and motion anticipation using burst-STDP." *PloS one*, vol. 7, no. 5, p. e36958, Jan. 2012.

[25] T. Serrano-Gotarredona, T. Masquelier, T. Prodromakis, G. Indiveri, and B. Linares-Barranco, "STDP and STDP variations with memristors for spiking neuromorphic learning systems." *Frontiers in neuroscience*, vol. 7, no. February, p. 2, Jan. 2013.

[26] G. S. Snider, R. Amerson, D. Carter, H. Abdalla, M. S. Qureshi, J. Leveille, M. Versace, H. Ames, S. Patrick, B. Chandler, A. Gorchetchnikov, and E. Mingolla, "From Synapses to Circuitry: Using Memristive Memory to Explore the Electronic Brain," *Computer*, vol. 44, no. 2, pp. 21–28, Feb. 2011.

[27] Z. Vasilkoski, H. Ames, B. Chandler, A. Gorchetchnikov, J. Leveille, G. Livitz, E. Mingolla, and M. Versace, "Review of stability properties of neural plasticity rules for implementation on memristive neuromorphic hardware," *The 2011 International Joint Conference on Neural Networks*, pp. 2563–2569, Jul. 2011.

[28] L. Chua, "Memristive devices and systems," *Proceedings of the IEEE*, vol. 64, no. 2, 1976.

[29] B. Widrow and M. Hoff, "Adaptive switching circuits," STANFORD ELECTRONICS LABS, Stanford, CA, Tech. Rep., 1960.

[30] B. Widrow and S. Stearns, *Adaptive signal processing*. Englewood Cliffs, NJ: Prentice-Hall, inc, 1985.

[31] L. Bottou and O. Bousquet, "The Tradeoffs of Large-Scale Learning," in *Optimization for Machine Learning*, 2011, p. 351.

[32] Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller, "Efficient backprop," in *Neural networks: Tricks of the Trade*, 2012.

[33] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based Material Implication (IMPLY) Logic: Design Principles and Methodologies," *submitted to IEEE Transactions on Very Large Scale Integration (VLSI)*, 2013.

[34] "See Supplemental Material."

[35] "http://www.mathworks.com/products/simulink/."

[36] "http://www.mathworks.com/products/simelectronics/."

[37] T. Chang, S.-H. Jo, K.-H. Kim, P. Sheridan, S. Gaba, and W. Lu, "Synaptic behaviors and modeling of a metal oxide memristive device," *Applied Physics A*, vol. 102, no. 4, pp. 857–863, Feb. 2011.

[38] A. C. Torrezan, J. P. Strachan, G. Medeiros-Ribeiro, and R. S. Williams, "Sub-nanosecond switching of a tantalum oxide memristor." *Nanotechnology*, vol. 22, no. 48, p. 485203, Dec. 2011.

[39] Y. LeCun and L. Bottou, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[40] "http://yann.lecun.com/exdb/mnist/."

[41] E. Oja, "Simplified neuron model as a principal component analyzer," *Journal of mathematical biology*, vol. 15, no. 3, pp. 267–273, 1982.

[42] R. Sarpeshkar, "Analog versus digital: extrapolating from electronics to neurobiology," *Neural Computation*, vol. 10, no. 7, pp. 1601–1638, 1998.

[43] "http://www.mosis.com."

[44] G. Huang, D. C. Sekar, A. Naeemi, K. Shakeri, and J. D. Meindl, "Compact Physical Models for Power Supply Noise and Chip/Package Co-Design of Gigascale Integration," in *2007 Proceedings 57th Electronic Components and Technology Conference*. Ieee, 2007, pp. 1659–1666.

[45] M. Hu, H. Li, Y. Chen, X. Wang, and R. E. Pino, "Geometry variations analysis of TiO2 thin-film and spintronic memristors," in *16th Asia and South Pacific Design Automation Conference (ASP-DAC 2011)*. Ieee, Jan. 2011, pp. 25–30.

[46] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 2, pp. 303–314, 1989.

[47] Y. Bengio, "Learning Deep Architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[48] D. Ciresan and U. Meier, "Deep, big, simple neural nets for handwritten digit recognition," *Neural Computation*, vol. 22, no. 12, pp. 3207–3220, 2010.

[49] C. M. Bishop, *Pattern recognition and machine learning*. Singapore: Springer, 2006.

[50] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter, "Pegasos: primal estimated sub-gradient solver for SVM," *Mathematical Programming*, vol. 127, no. 1, pp. 3–30, Oct. 2010.

[51] L. Chua, "Resistance switching memories are memristors," *Applied Physics A*, vol. 102, no. 4, pp. 765–783, 2011.

[52] D. B. Strukov and K. K. Likharev, "Reconfigurable nano-crossbar architectures," in *Nanoelectronics and Information Technology*, 2012, pp. 543–562.

[53] R. Legenstein, C. Naeger, and W. Maass, "What can a neuron learn with spike-timing-dependent plasticity?" *Neural computation*, vol. 17, no. 11, pp. 2337–82, 2005.

[54] G. Cauwenberghs, "Analysis and verification of an analog VLSI incremental outer-product learning system," *IEEE transactions on neural networks*, 1992.

[55] H. Card, C. R. Schneider, and W. R. Moore, "Hebbian plasticity in mos synapses," *IEEE Transactions on Audio and Electroacoustics*, vol. 138, no. 1, p. 13, 1991.

[56] C. Schneider and H. Card, "Analogue CMOS Hebbian Synapses," *Electronics letters*, pp. 1990–1991, 1991.

[57] H. Card, C. R. Schneider, and R. S. Schneider, "Learning capacitive weights in analog CMOS neural networks," *Journal of VLSI Signal Processing*, vol. 8, no. 3, pp. 209–225, Oct. 1994.

[58] 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, 1996.

[59] 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, 1994.

[60] C. Lu, B. Shi, and L. Chen, "An on-chip BP learning neural network with ideal neuron characteristics and learning rate adaptation," *Analog Integrated Circuits and Signal Processing*, pp. 55–62, 2002.

[61] 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, 1992.

[62] C. S. Lindsey and T. Lindblad, "Survey of neural network hardware," in *Proc. SPIE, Applications and Science of Artificial Neural Networks*, vol. 2492, 1995, pp. 1194–1205.

[63] G. Cauwenberghs, "A learning analog neural network chip with continuous-time recurrent dynamics," *NIPS*, 1994.

[64] H. Eguchi, T. Furuta, and H. Horiguchi, "Neural network LSI chip with on-chip learning," *Neural Networks*, pp. 453–456, 1991.

[65] C. Schneider and H. Card, "CMOS implementation of analog Hebbian synaptic learning circuits," *IJCNN-91-Seattle International Joint Conference on Neural Networks*, vol. i, pp. 437–442, 1991.