Memristor-Based Multilayer Neural Networks With Online Gradient Descent Training

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

***Abstract*— Learning in multilayer neural networks (MNNs) relies on continuous updating of large matrices of synaptic weights by local rules. Such locality can be exploited for massive parallelism when implementing MNNs in hardware. However, these update rules require a multiply and accumulate operation for each synaptic weight, which is challenging to implement compactly using CMOS. In this paper, a method for performing these update operations simultaneously (incremental outer products) using memristor-based arrays is proposed. The method is 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% and 8% of the area and static power of previous CMOS-only hardware alternatives. Such a circuit can compactly implement hardware MNNs trainable by scalable algorithms based on online gradient descent (e.g., backpropagation). The utility and robustness of the proposed memristor-based circuit are demonstrated on standard supervised learning tasks.**

***Index Terms*— Backpropagation, hardware, memristive systems, memristor, multilayer neural networks (MNNs), stochastic gradient descent, synapse.**

I. Introduction

MULTILAYER neural networks (MNNs) have been recently incorporated into numerous commercial products and services such as mobile devices and cloud computing. For realistic large scale learning tasks, MNNs can perform impressively well and produce state-of-the-art results when massive computational power is available [1]–[3] (see [4] for related press). However, such computational intensity limits their usability due to area and power requirements. New dedicated hardware design approach must therefore be developed to overcome these limitations. It was recently suggested that such new types of specialized hardware are essential for real progress toward building intelligent machines [5].

MNNs 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 MNNs 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*. A canonical example for a local learning rule is the backpropagation algorithm, an efficient implementation of online gradient descent, which is commonly used to train MNNs [6]. The locality of backpropagation stems from the chain rule used to calculate the gradients. Similar locality appears in many other learning rules used to train neural networks and various machine learning (ML) algorithms.

Implementing ML algorithms such as backpropagation on conventional general-purpose digital hardware (i.e., von Neumann architecture) is highly inefficient. A primary 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 these learning rules—their locality. This locality allows highly efficient parallel computation, as demonstrated by biological brains.

To overcome the inefficiency of general-purpose hardware, numerous dedicated hardware designs, based on CMOS technology, have been proposed in the past two decades [7, and references therein]. These designs perform online learning tasks in MNNs using 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 applications, and it is not clear whether they could be scaled up, since each synapse requires too much power and area (Section VII). This issue of scalability possibly casts doubt on the entire field [8].

Recently, it has been suggested [9]–[10] that scalable hardware implementations of neural networks may become possible if a novel device, the *memristor* [11]–[14], is used. A memristor is a resistor with a varying history-dependent resistance. It is a passive analog device with

Manuscript received June 16, 2014; revised December 5, 2014; accepted December 11, 2014. Date of publication January 14, 2015; date of current version September 16, 2015. This work was supported in part by the Gruss Lipper Charitable Foundation, in part by the Intel Collaborative Research Institute for Computational Intelligence, in part by the Hasso Plattner Institute, Potsdam, Germany, and in part by the Andrew and Erna Finci Viterbi Fellowship Program.

D. Soudry is with the Center for Theoretical Neuroscience, and the Grossman Center for the Statistics of Mind, Department of Statistics, Columbia University, New York, NY 10027 USA (e-mail: daniel.soudry@gmail.com).

D. Di Castro is with Yahoo! Labs, Haifa 31905, Israel (e-mail: dotan.dicastro@gmail.com).

A. Gal is with the Department of Electrical Engineering, Biological Networks Research Laboratories, Technion-Israel Institute of Technology, Haifa 32000, Israel (e-mail: asafg1@gmail.com).

A. Kolodny is with the Department of Electrical Engineering, Technion-Israel Institute of Technology, Haifa 32000, Israel (e-mail: kolodny@ee.technion.ac.il).

S. Kvatinsky is with the Department of Computer Science, Stanford University, Stanford, CA 94305 USA (e-mail: skva@tx.technion.ac.il).

This paper has supplemental material available online at http://ieeexplore.ieee.org (File size: 9 MB).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2014.2383395

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 [15] and allows a compact and efficient architecture for learning algorithms, as well as other neural-network-related applications ([16]–[18], and references therein), which will not be discussed here.

Most previous implementations of learning rules using memristor arrays have been limited to spiking neurons and focused on spike-timing-dependent plasticity (STDP) [19]–[26]. Applications of STDP are usually aimed to explain biological neuroscience results. At this point, however, it is not clear how useful is the STDP algorithmically. For example, the convergence of STDP-based learning is not guaranteed for general inputs [27]. Other learning systems [28]–[30] that rely on memristor arrays are limited to single-layer neural networks (SNNs) with a few inputs per neuron. To the best of the authors' knowledge, the learning rules used in the above memristor-based designs are not yet competitive and have not been used for large-scale problems, which is precisely where dedicated hardware is required.

A recent memristor array design [31] implemented the scalable perceptron algorithm [32]. This algorithm can be potentially used to train SNNs with binary outputs on very large datasets. However, training general MNNs (which are much more powerful than SNNs [33]) cannot be performed using the perceptron algorithm. Scalable training of MNNs requires the backpropagation algorithm. This is special form of online gradient descent, which is generally very effective in large-scale problems [34]. Importantly, it can achieve state-of-the-art-results on large datasets when executed with massive computational power [3], [35].

To date, no circuit has been suggested to utilize memristors for implementing such scalable online learning in MNNs. Interestingly, it was recently shown [36], [37] that memristors could be used to implement MNNs trained by backpropagation in a chip-in-a-loop setting, where the weight update calculation is performed using a host computer. However, it remained an open question whether the learning itself, which is a major computational bottleneck, could also be done *online* (i.e., completely implemented using hardware) with efficient massively parallel memristor arrays. The main challenge for general synaptic array circuit design arises from the nature of learning rules such as backpropagation: practically, all of them contain a multiplicative term [38], which is hard to implement in compact and scalable hardware.

In this paper, a novel and general scheme to design hardware for online gradient descent 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, therefore, requires 2%–8% of the area and static power of the previously proposed CMOS-only circuits.

Using this proposed scheme, for the first time, it is possible to implement a memristor-based hardware MNN capable of online learning (using the scalable backpropgation algorithm). The functionality of such a hardware MNN circuit utilizing the memristive synapse array is demonstrated numerically on standard supervised learning tasks. On all datasets, the circuit performs as well as the software algorithm. Introducing noise levels of about 10% and parameter variability of about 30% only mildly reduced the performance of the circuit, due to the inherent robustness of online gradient descent (Fig. 5). The proposed design may therefore allow the use of specialized hardware for MNNs, as well as other ML algorithms, rather than the currently used general-purpose architecture.

The remainder of this paper is organized as follows. In Section II, a basic background on memristors and online gradient descent learning is given. In Section III, the proposed circuit is described for efficient implementation of SNNs in hardware. In Section IV, a modification of the proposed circuit is used to implement a general MNN. In Section V, the sources of noise and variation in the circuit are estimated. In Section VI, the circuit operation and learning capabilities are evaluated numerically, demonstrating that it can be used to implement MNNs trainable with online gradient descent. In Section VII, the novelty of the proposed circuit is discussed, as well as possible generalizations, and in Section VIII, this paper is summarized. The supplementary material [39] includes detailed circuit schematics, code, and an appendix with additional technicalities.

II. Preliminaries

For convenience, a basic background information on memristors and online gradient descent learning is given in this section. For simplicity, the second part is focused on a simple example of the adaline algorithm—a linear SNN trained using mean square error (MSE).

A. Memristor

The memristor was originally proposed [11], [12] as the missing fourth fundamental passive circuit 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:

$$i(t) = G(s(t))v(t) \tag{1}$$

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

A generalization of the memristor model 1, 2, which is called a *memristive system*, was proposed in [40]. In memristive devices, s is a general state variable, rather than an integral of the voltage. Such memristive models, which are more commonly used to model actual physical devices [14], [41], [42], are discussed in [Appendix A, 39].

For the sake of generality and simplicity, in the following sections, it is assumed that the variations in the value of $s(t)$ are restricted to be small so that $G(s(t))$ can be linearized around some point s^*, and the conductivity of the memristor is given, to first order, by

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

where $\hat{g} = [dG(s)/ds]_{s=s^*}$ and $\bar{g} = G(s^*) - \hat{g}s^*$. Such a linearization is formally justified if sufficiently small inputs are used, so s does not stray far from the fixed point (i.e., $2\hat{g}/[d^2G(s)/ds^2]_{s=s^*} \gg |s(t)-s_*|$, so second-order contributions are negligible). The operation in such a small signal region is demonstrated numerically in [Appendix D, 39], for a family of physical memristive device technologies. The only (rather mild) assumption is that $G(s)$ is differentiable near s^*.

Note that despite this linearization, the memristor is still a nonlinear component, since [from (1) and (3)] $i(t) = \bar{g}v(t) + \hat{g}s(t)v(t)$. Importantly, this nonlinear product $s(t)v(t)$ underlies the key role of the memristor in the proposed design, where an input signal $v(t)$ is being multiplied by an *adjustable* internal value $s(t)$. Thus, the memristor enables an efficient implementation of trainable MNNs in hardware, as explained below.

B. Online Gradient Descent Learning

The field of ML is dedicated to the construction and study of systems that can be learned from data. For example, consider the following *supervised learning* task. Assume a learning 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 $\mathbf{d}^{(k)} = \mathbf{f}(\mathbf{x}^{(k)})$. Note that two distinct patterns can have the same label. The objective of the system is to estimate (learn) the function $\mathbf{f}(\cdot)$ using the empirical data.

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

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

or

$$r_n^{(k)} = \sum_m W_{nm}^{(k)} x_m^{(k)} \tag{5}$$

which is a SNN. The *result* of the estimator $\mathbf{r} = \mathbf{W}\mathbf{x}$ should aim to predict the right *desired* labels $\mathbf{d} = \mathbf{f}(\mathbf{x})$ for new unseen *patterns* $\mathbf{x}$. To solve this problem, $\mathbf{W}$ is tuned to minimize some measure of error between the estimated and desired labels, over a K_0-long subset of the empirical data, called the training set (for which $k = 1, \ldots, K_0$). 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 MSE

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

Other error measures can be also be used. The performance of the resulting estimator is then tested over a different subset, called the test set ($k = K_0 + 1, \ldots, K$).

As explained in the introduction, 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:

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

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 $\mathbf{x}^{(k)}$ is chosen randomly and presented at the input of the system. Using the chain rule (4), (6), we have $\nabla_{\mathbf{W}^{(k)}} \|\mathbf{y}^{(k)}\|^2 = -2(\mathbf{d}^{(k)} - \mathbf{W}^{(k)}\mathbf{x}^{(k)})(\mathbf{x}^{(k)})^\top$. Therefore, defining $\Delta\mathbf{W}^{(k)} \triangleq \mathbf{W}^{(k+1)} - \mathbf{W}^{(k)}$ and $(\cdot)^\top$ to be the transpose operation, we obtain the *outer product*

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

or

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

Specifically, this update rule is called the adaline algorithm [43], used in adaptive signal processing and control [44]. The parameters of more complicated estimators can also be similarly tuned (trained), using online gradient descent or similar methods. Specifically, MNNs (Section IV) are commonly being trained using backpropagation, which is an efficient form of online gradient descent [6]. Importantly, note that the update rule in (10) is local, i.e., the change in the synaptic weight $W_{nm}^{(k)}$ depends only on the related components of input ($x_m^{(k)}$) and error ($y_n^{(k)}$). This local update, which ubiquitously appears in neural network training (e.g., backpropagation and the perceptron learning rule [32]) and other ML algorithms [45]–[47], enables a massively parallel hardware design, as explained in Section III.

Such massively parallel designs are needed, since for large N and M, learning systems usually become computationally prohibitive in both time and memory space. For example, in the simple adaline algorithm, the main computational burden in each iteration comes from (4) and (9), 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 MNNs (and related ML algorithms) in software. Other algorithmic steps, such as (6) here, include either $O(M)$ or $O(N)$ operations and, therefore, have a negligible computational complexity, lower than $O(M + N)$.

III. Circuit Design

Next, dedicated analog hardware for implementing online gradient descent learning is described. For simplicity, this section is focused on simple linear SNNs trained using adaline, as described in Section II. Later, in Section IV, the circuit is modified to implement general MNNs, trained using backpropagation. The derivations are rigorously done using a single controlled approximation (14) and no heuristics (i.e., unproven methods which can damage performance), thus creating a precisely defined mapping between a mathematical learning system and a hardware learning system. Readers who do not wish to go through the detailed derivations, can get the general

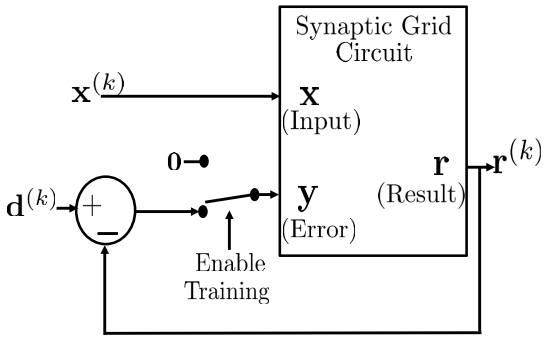

Fig. 1. Simple adaline learning task, with the proposed synaptic grid circuit executing (4) and (9), which are the main computational bottlenecks in the algorithm.

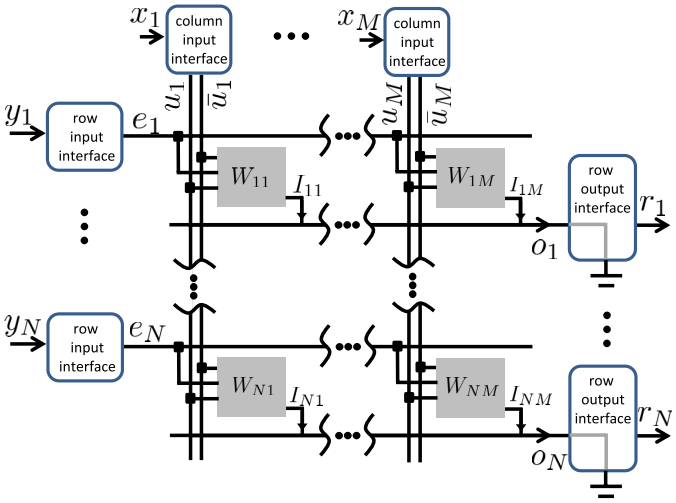

Fig. 2. Synaptic grid ($N \times M$) circuit architecture scheme. Every (n, m) node in the grid is a memristor-based synapse that receives voltage input from the shared $u_m, \bar{u}_m$ and the e_n lines and outputs I_{nm} current on the o_n lines. These output lines receive total current arriving from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

idea from the following overview (Section III-A) together with Figs. 1–4.

A. Circuit Overview

To implement these learning systems, a grid of artificial synapses is constructed, where each synapse stores 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×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×vector outer product in (10) is done 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 auxiliary circuits that handle a negligible amount of $O(M + N)$ additional operations, these arrays can be used to construct efficient learning systems. These systems perform

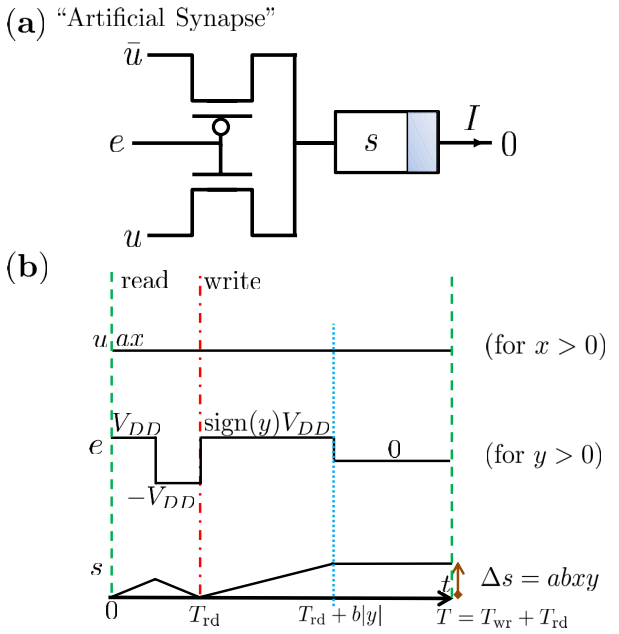

Fig. 3. Memristor-based synapse. (a) Schematic of a single memristive synapse (without the n and m indices). The synapse receives input voltages u and $\bar{u} = -u$, an enable signal e, and output current I. (b) Read and write protocols—incoming signals in a single synapse and the increments in the synaptic weight s, as determined by (27). $T = T_{\rm wr} + T_{\rm rd}$.

massive parallelization of the bottleneck $O(N \cdot M)$ operations (4) and (9) over many computational units—the synapses.

Similarly to the adaline algorithm described in Section II, the circuit operates on discrete presentations of inputs (trials) (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 (4), where the matrix $\mathbf{W}^{(k)} \in \mathbb{R}^{N \times M}$, called the *synaptic weight matrix*, is stored in the system. In addition, on each step, the circuit updates $\mathbf{W}^{(k)}$ according to (9). This circuit can be used to implement ML algorithms. For example, as shown 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 the backpropagation algorithm is shown in Fig. 4 and explained in Section IV.

B. Circuit Architecture

1) Synaptic Grid: The synaptic grid system described in Fig. 1 is implemented by the circuit shown in Fig. 2, where the 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. 3(a)]. The synapses are arranged in a 2-D $N \times M$ grid array as shown in Fig. 2, where each synapse is indexed by (n, m), with $m \in \{1, \ldots, M\}$ and $n \in \{1, \ldots, N\}$. Each (n, m) synapse receives two *inputs* $u_m, \overline{u}_m$, an *enable* signal e_n, and produces an *output* current I_{nm}. Each column of synapses in the array (the mth 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.

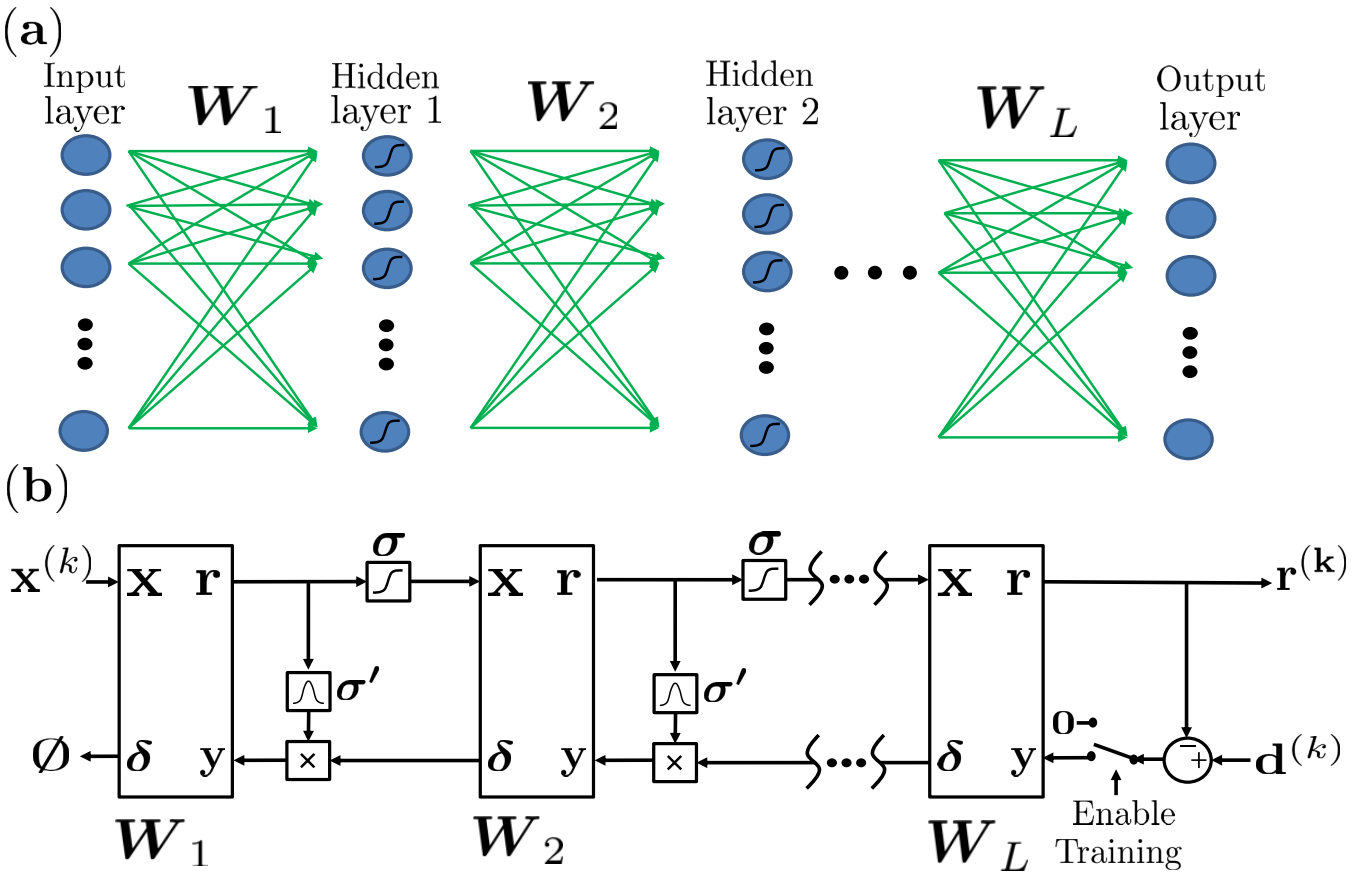

Fig. 4. (a) Structure of a MNN with the neurons in each layer (circles) and the synaptic connectivity weights (arrows). (b) Implementation of a MNN with on-line backpropagation training with MSE measure (7), using a slightly modified version of the original circuit (Fig. 2). Each circuit performs (23) and (28) (as in Fig. 1), together with the additional operation (31). The function boxes denote the operation of either $\sigma(\cdot)$ (neuronal activation function) or $\sigma'(\cdot)$ (its derivative) on the input, and the $\times$ box denotes a component-wise product. The symbol $\emptyset$ (don't care) denotes an unused output. For detailed circuit schematics, see [39].

Each row of synapses in the array (the nth 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 o_n lines grounded (to zero voltage) and convert the total current from all the synapses in the row going to the ground, $\sum_m I_{nm}$, to the output signal r_n.

2) 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 Fig. 3(a) (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 that 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. In addition, the current of the n-type transistor in the linear region is, ideally

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

where $V_{\rm GS}$ is the gate–source voltage, V_T is the threshold voltage, $V_{\rm DS}$ is the drain–source voltage, and K is the conduction parameter of the transistors. When $V_{\rm GS} < V_T$, the current is cutoff ($I = 0$). Similarly, the current of the p-type transistor in the linear region is

$$I = -K\left((V_{\rm GS} + V_T)V_{\rm DS} - \frac{1}{2}V_{\rm DS}^2\right) \tag{12}$$

where for simplicity, we assumed that the parameters K and V_T are equal for both transistors. Note that for notational simplicity, the parameter V_T in (12) has a different sign than in the usual definition. When $V_{\rm GS} > -V_T$, the current is cutoff ($I = 0$).

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, $V_{\rm DD}$, or $-V_{\rm DD}$ (with $V_{\rm DD} > V_T$) 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.

1) We assume (somewhat unusually) that

$$-V_T < u(t) < V_T. \tag{13}$$

2) We assume that [recall (1)]

$$K(V_{\rm DD} - 2V_T) \gg G(s(t)). \tag{14}$$

From the first assumption (13), we assume the following conditions.

1) If $e = 0$ (i.e., the gate is grounded), both transistors are nonconducting (in the cutoff region). In this case, $I = 0$ in the output, the voltage across the memristor is zero, and the state variable does not change.
2) If $e = V_{\mathrm{DD}}$, the n-type transistor is conducting in the linear region while the p-type transistor is nonconducting.
3) If $e = -V_{\mathrm{DD}}$, the p-type transistor is conducting in the linear region while the n-type transistor is nonconducting.

To satisfy (13), the proper value of $u(t)$ is chosen (15).

From the second assumption, (14), if $e = \pm V_{\mathrm{DD}}$, when in the linear region, both transistors have relatively high conductivity as compared with the conductivity of the memristor. Therefore, in that case, the voltage on the memristor is approximately $\pm u$. Note (14) is a reasonable assumption, as shown in [48]. If not (e.g., if the memristor conductivity is very high), instead one can use an alternative design, as described in [Appendix B, 39].

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 $\Delta\mathbf{W} = \eta\mathbf{y}\mathbf{x}^{\top}$ from (9). 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. 3(b).

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_{rd}, $\forall m : u_m(t) = ax_m = -\bar{u}_m(t)$, where a is a positive constant converting x_m, a unitless number, to voltage. Recall that A is the maximal value of $|x_m|$, so

$$aA < V_T \tag{15}$$

as required in (13). In addition, the row input interfaces produce voltage signal on the e_n lines, $\forall n$

$$e_n(t) = \begin{cases} V_{\mathrm{DD}}, & \text{if } 0 \le t < 0.5T_{\mathrm{rd}} \\ -V_{\mathrm{DD}}, & \text{if } 0.5T_{\mathrm{rd}} \le t \le T_{\mathrm{rd}}. \end{cases} \tag{16}$$

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

$$\Delta s_{nm} = \int_0^{0.5T_{\mathrm{rd}}} (ax_m)dt + \int_{0.5T_{\mathrm{rd}}}^{T_{\mathrm{rd}}} (-ax_m)dt = 0. \tag{17}$$

The zero net change in the value of s_{nm} between the times of 0 and T_{rd} implements a nondestructive read, as common in many memory devices. To minimize inaccuracies, the row output interface samples the output current at time 0^+ (immediately after time zero). This is done before the conductance of the memristor is significantly changed from its value before the read phase. Using (3), the output current of the synapse to the o_n line at the time is thus

$$I_{nm} = a(\bar{g} + \hat{g}s_{nm})x_m. \tag{18}$$

Therefore, the total current in each output line o_n is equal to the sum of the individual currents produced by the synapses driving that line

$$o_n = \sum_m I_{nm} = a\sum_m (\bar{g} + \hat{g}s_{nm})x_m. \tag{19}$$

The row output interface measures the output current o_n and outputs

$$r_n = c(o_n - o_{\mathrm{ref}}) \tag{20}$$

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

$$o_{\mathrm{ref}} = a\bar{g}\sum_m x_m. \tag{21}$$

Defining

$$W_{nm} = ac\hat{g}s_{nm} \tag{22}$$

we obtain

$$\mathbf{r} = \mathbf{W}\mathbf{x} \tag{23}$$

as desired.

2) Update Phase (Write): During each write phase, of duration of T_{wr}, $\mathbf{u}$ and $\bar{\mathbf{u}}$ maintain their values from the read phase, while the signal $\mathbf{e}$ changes. In this phase, the row input interfaces encode $\mathbf{e}$ componentwise, $\forall n$

$$e_n(t) = \begin{cases} \mathrm{sign}(y_n)V_{\mathrm{DD}}, & \text{if } 0 \le t - T_{\mathrm{rd}} \le b|y_n| \\ 0, & \text{if } b|y_n| < t - T_{\mathrm{rd}} < T_{\mathrm{wr}}. \end{cases} \tag{24}$$

The interpretation of (24) is that e_n is a pulse with a magnitude V_{DD}, the same sign as y_n, and a duration $b|y_n|$ (where b is a constant converting y_n, a unitless number, to time units). Recall that A is the maximal value of $|y_n|$, so we require that

$$T_{\mathrm{wr}} > bA. \tag{25}$$

The total change in the internal state is therefore

$$\Delta s_{nm} = \int_{T_{\mathrm{rd}}}^{T_{\mathrm{rd}}+b|y_n|} (a\,\mathrm{sign}(y_n)x_m)dt \tag{26}$$

$$= abx_m y_n. \tag{27}$$

Using (22), the desired update rule for the synaptic weights is, therefore, obtained as

$$\Delta\mathbf{W} = \eta\mathbf{y}\mathbf{x}^{\top} \tag{28}$$

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

IV. Multilayer Neural Network Circuit

So far, the circuit operation was exemplified on a SNN with the simple adaline algorithm. In this section, it is explained how, with a few adjustments, the proposed circuit can be used to implement backpropagation on a general MNN.

Recall the context of the supervised learning setup detailed in Section II, where $\mathbf{x} \in \mathbb{R}^M$ is a given pattern and $\mathbf{d} \in \mathbb{R}^N$ is a desired label. Consider a *double layer MNN* estimator of $\mathbf{d}$, of the form

$$\mathbf{r} = \mathbf{W}_2 \sigma(\mathbf{W}_1 \mathbf{x})$$

with $\mathbf{W}_1 \in \mathbb{R}^{H \times M}$ (H is some integer) and $\mathbf{W}_2 \in \mathbb{R}^{N \times H}$ being the two parameter matrices and where σ is some nonlinear (usually sigmoid) function operating component wise [i.e., $(\sigma(\mathbf{x}))_i = \sigma(x_i)$]. Such a double layer MNN, with H hidden neurons, can approximate any target function with arbitrary precision [33]. Denote by

$$\mathbf{r}_1 = \mathbf{W}_1 \mathbf{x} \in \mathbb{R}^H; \ \mathbf{r}_2 = \mathbf{r} = \mathbf{W}_2 \sigma(\mathbf{r}_1) \in \mathbb{R}^N \tag{29}$$

the output of each layer. Suppose we again use MSE to quantify the error between $\mathbf{d}$ and $\mathbf{r}$. In the backpropagation algorithm, each update of the parameter matrices is given by an online gradient descent step, which can be directly derived from [6, eq. (8)] [note the similarity with (9)]

$$\Delta \mathbf{W}_1 = \eta \mathbf{y}_1 \mathbf{x}_1^\top; \ \Delta \mathbf{W}_2 = \eta \mathbf{y}_2 \mathbf{x}_2^\top \tag{30}$$

with $\mathbf{x}_2 \triangleq \sigma(\mathbf{r}_1)$, $\mathbf{y}_2 \triangleq \mathbf{d} - \mathbf{r}_2$, $\mathbf{x}_1 \triangleq \mathbf{x}$ and $\mathbf{y}_1 \triangleq (\mathbf{W}_2^\top \mathbf{y}_2) \times \sigma'(\mathbf{r}_1)$, where here $(\mathbf{a} \times \mathbf{b})_i \triangleq a_i b_i$, a component-wise product, and $(\sigma'(\mathbf{x}))_i = d\sigma(x_i)/dx_i$. Implementing such an algorithm requires a minor modification of the proposed circuit, that is, it should have an additional inverted output $\mathbf{W}^\top \mathbf{y}$. 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. 4. For each layer, a synaptic grid circuit stores the corresponding weight matrix, performs a matrix-vector product as in (29), and updates the weight matrix as in (30). This last operation requires the additional output

$$\boldsymbol{\delta} \triangleq \mathbf{W}^\top \mathbf{y} \tag{31}$$

in each synaptic grid circuit (except the first). We assume that the synaptic circuit has dimensions $M \times N$ (with a slight abuse of notation, since M and N should be different for each layer). The additional output (31) can be generated by the circuit in an additional read phase with the duration T_{rd}, between the original read and write phases, in which the original role of the input and output lines is inverted. In this phase, the n-type MOS (nMOS) transistor is ON, i.e., $\forall n$, $e_n = V_{\mathrm{DD}}$ and the former output o_n lines are given the following voltage signal (again, used for a nondestructive read):

$$o_n = \begin{cases} a y_n, & \text{if } T_{\mathrm{rd}} \le t < 1.5 T_{\mathrm{rd}} \\ -a y_n, & \text{if } 1.5 T_{\mathrm{rd}} \le t \le 2 T_{\mathrm{rd}}. \end{cases} \tag{32}$$

The I_{nm} current now flows to the (original input) u_m terminal, which is grounded and shared $\forall n$. The sum of all the currents is measured at time T_{rd}^+ by the column interface (before it goes into ground)

$$u_m = \sum_n I_{nm} = a \sum_n (\bar{g} + \hat{g} s_{nm}) y_n. \tag{33}$$

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

$$\delta_m = c(u_m - u_{\mathrm{ref}}) \tag{34}$$

where $u_{\mathrm{ref}} = a\bar{g} \sum_n y_n$. Thus, from (22)

$$\delta_m = \sum_n W_{nm} y_n \tag{35}$$

as required.

Finally, note that it is straightforward to use a different error function instead of MSE. For example, in a MNN, recall that in the last layer

$$\mathbf{y} = \mathbf{d} - \mathbf{r} = -\frac{1}{2} \nabla_{\mathbf{r}} \ \|\mathbf{d} - \mathbf{r}\|^2.$$

If a different error function $E(\mathbf{r}, \mathbf{d})$ is required, one should simply replace this with

$$\mathbf{y} \triangleq -\alpha \nabla_{\mathbf{r}} E(\mathbf{r}, \mathbf{d}) \tag{36}$$

where α is some constant that can be used to tune the learning rate. For example, one could use instead a cross entropy error function

$$E(\mathbf{r}, \mathbf{d}) = -\sum_i d_i \ln r_i \tag{37}$$

which is more effective for classification tasks, in which $\mathbf{d}$ is a binary vector [49]. To implement this change in the MNN circuit (Fig. 4), the subtractor should be simply replaced with some other module.

V. Sources of Noise and Variability

Usually, analog computation suffers from reduced robustness to noise as compared with digital computation [50]. ML algorithms are, however, inherently robust to noise, which is a key element in the set of problems they are designed to solve. For example, gradient descent is quite robust to perturbations, as intuitively demonstrated in Fig. 5. 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 implementation (e.g., the CMOS process). Particularly, memristor technology is not mature yet and memristors have not been fully characterized. Therefore, to check the robustness of the circuit, crude estimation of the magnitude of noise and variability has been used in this section. This estimation is based on known sources of noise and variability, which are less dependent on the specific implementation. In Section VI, the alleged robustness of the circuit would be evaluated by simulating the circuit in the presence of these noise and variation sources.

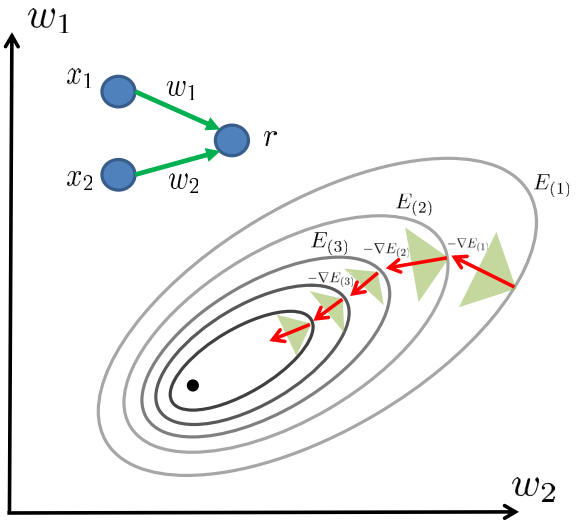

Fig. 5. Robustness of gradient descent. Consider a simple $2 \to 1$ SNN (inset) with a single input $\mathbf{x}$ and desired output d (repeatedly presented). As demonstrated schematically in this figure, training the SNN using gradient descent on the MSE $E(w_1, w_2) = (w_1 x_1 + w_2 x_2 - d)^2$ will tend to decrease this error until it converges to some fixed point (generally, a local minimum of the MSE). Schematically, the gradient direction (red arrows) can be arbitrarily varied within a relatively wide range (green triangles), on each iteration, and this will not prevent this convergence.

A. Noise

When one of the transistors is enabled ($e(t) = \pm V_{\mathrm{DD}}$), then the current is affected by intrinsic thermal noise sources in the transistors and memristor of each synapse. This noise affects the operation of the circuit during the write and read phases. 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 $\sigma^2 = 2\ \mathrm{k}\tilde{T}\mathrm{g}$ (where k is Boltzmann's constant, $\tilde{T}$ is the temperature, and g is the conductance of the device). For 65 nm transistors (parameters taken from IBMs 10LPe/10RFe process [51]), the characteristic conductivity is $g_1 \sim 10^{-4}\ \Omega^{-1}$. Therefore, for $I_1(t)$, the thermal current source of the transistors at room temperature, we have $\sigma_1^2 \sim 10^{-24}\ \mathrm{A}^2\ \mathrm{s}$. 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 (14), we have $\epsilon \ll 1$, and 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 $\xi(t) = (g_1^{-1}/1 + \epsilon)(I_1(t) - I_2(t))$. Since different thermal noise sources are uncorrelated, we have $\langle I_1(t) I_2(t') \rangle = 0$, and so $\langle \xi(t) \xi(t') \rangle \sim \sigma_\xi^2 \delta(t - t')$ with

$$\begin{aligned} \sigma_\xi^2 &= \frac{1}{g_1^2 (1 + \epsilon)^2} \left(\langle I_1^2(t) \rangle + \langle I_2^2(t) \rangle \right) \\ &= \frac{\sigma_1^2 + \sigma_2^2}{g_1^2 (1 + \epsilon)^2} \\ &\approx g_1^{-2} \sigma_1^2 = 2k\tilde{T} g_1^{-1} \sim 10^{-16}\ \mathrm{V}^2\mathrm{s}. \end{aligned} \tag{38}$$

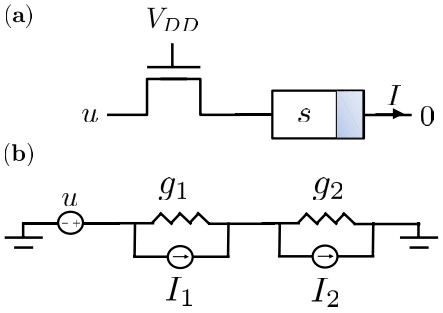

Fig. 6. Noise model for an artificial synapse. (a) During the operation, only one transistor is conducting (assume it is the n-type transistor). (b) Thermal noise in a small-signal model: the transistor is converted to a resistor (g_1) in parallel to current source (I_1), the memristor is converted to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

The equivalent circuit, including the sources of noise, is shown in Fig. 6. Assuming the circuit minimal trial duration is $T = 10$ ns, the root MSE due to thermal noise is bounded above by

$$\begin{aligned} E_T &\sim \sqrt{\left\langle \left(\frac{1}{T} \int_0^T d\xi(t) \right)^2 \right\rangle} \\ &= T^{-1/2} \sigma_\xi \sim 10^{-4} V. \end{aligned}$$

Noise in the inputs $u, \bar{u}$, and e also exists. According to [52], 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 $|E_u| \leq 0.1a|x|$. The absolute noise level in duration of e should be smaller than $T_{\mathrm{clk}}^{\min} \sim 2 \cdot 10^{-10}$ s, assuming a digital implementation of pulsewidth modulation with $T_{\mathrm{clk}}^{\min}$ being the shortest clock cycle currently available. On every write cycle $e = \pm V_{\mathrm{DD}}$ is, therefore, applied for a duration of $b|y| + E_e$ (instead of $b|y|$), where $|E_e| < T_{\mathrm{clk}}^{\min}$.

B. Parameter Variability

A common estimation of the variability in memristor parameters is a coefficient of variation (CV = standard deviation/mean) of a few percent [53]. In this paper, the circuit is also evaluated with considerably larger variability (CV $\sim$ 30%), in addition to the noise sources, as described in Section V-A. The variability in the parameters of the memristors is modeled by sampling each memristor conductance parameter $\hat{g}$ independently from a uniform distribution between 0.5 and 1.5 of the original (nonrandom) value of $\hat{g}$. When running the algorithm in software, these variations are equivalent to corresponding changes in the synaptic weights W or the learning rate η in the algorithm. Note that the variability in the transistor parameters is not considered, since these can affect the circuit operation only if (13) or (14) are invalidated. This can happen, however, only if the values of K or V_T vary in orders of magnitude, which is unlikely.

VI. Circuit Evaluation

In this section, the proposed synaptic grid circuit is implemented in a simulated physical model (Section VI-A).

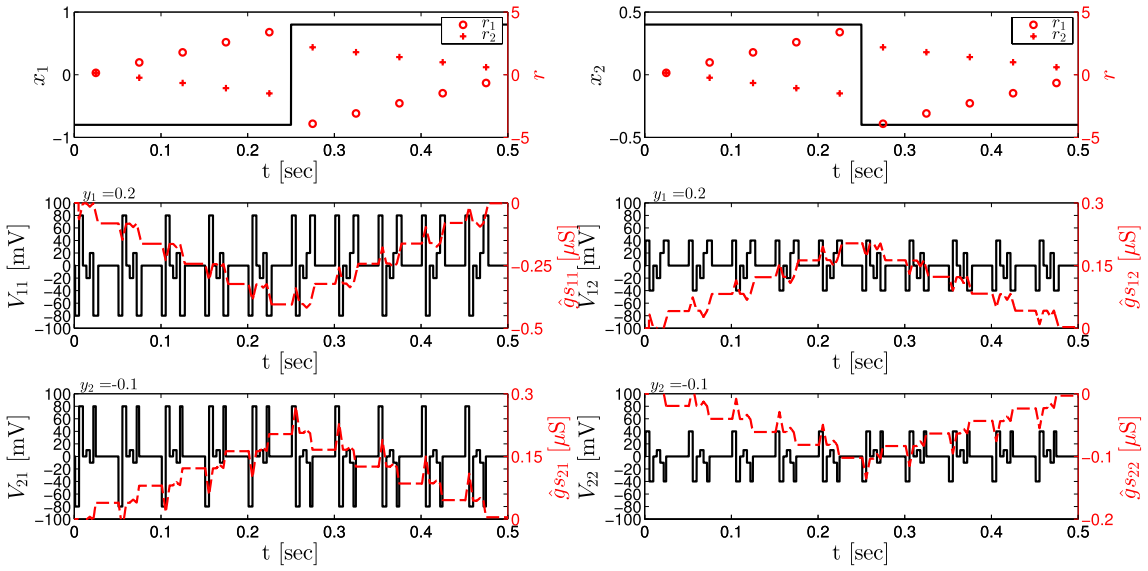

Fig. 7. Synaptic 2×2 grid circuit simulation, during ten operation cycles. Top: circuit inputs (x_1, x_2) and result outputs (r_1, r_2). Middle and bottom: voltage (solid black) and conductance (dashed red) change for each memristor in the grid. Note that the conductance of the (n, m) memristor indeed changes proportionally to $x_m y_n$ following (27). Simulation was done using SimElectronics, with the inputs as in (39) and (40), and the circuit parameters as in Table I. Similar results were obtained using SPICE with linear ion drift memristor model and CMOS 0.18 μm process, as shown in [Appendix D, 39].

First, the basic functionality of a toy example, a 2×2 circuit, is demonstrated (Section VI-B). Then (Section VI-C) using standard supervised learning datasets, the implementation of SNNs and MNNs using this synaptic grid circuit is demonstrated, including its robustness to noise and variation.

A. Software Implementation

Recall that the synaptic grid circuit (implementing the boxes in Figs. 1 and 4) operates in discrete time trials, and receives at each trial two vector inputs $\mathbf{x}$ and $\mathbf{y}$, updates an internally stored synaptic matrix $\mathbf{W}$ (according to $\Delta\mathbf{W} = \eta\mathbf{y}\mathbf{x}^\top$), and outputs the vector $\mathbf{r} = \mathbf{W}\mathbf{x}$ (optionally, it outputs also the vector $\boldsymbol{\delta} = \mathbf{W}^\top\mathbf{y}$).

The physical model of the circuit is implemented both in SPICE and SimElectronics [54]. Both are software tools that enable physical circuit simulation of memristors and MOSFET devices. The SPICE model is described in [Appendix D, 39].

Next, the SimElectronics synaptic grid circuit model is described. Exactly the same model (at different sizes of grid) is used for all numerical evaluations in Sections VI-C. The circuit parameters appear in Table I, with the parameters of the (ideal) transistors kept at their defaults. Note V_T of the pMOS is defined here with an opposite sign to the usual definition. The memristor model is implemented using (1)–(3) with parameters taken from the experimental data [55, Fig. 2]. As shown schematically in Fig. 2, the circuit implementation consists of a synapse-grid and the interface blocks. The interface units were implemented using a few standard CMOS and logic components, as can be seen in the detailed schematics (available in [39] as an HTML file, which may be opened using Firefox). The circuit operated synchronously using global control signals supplied externally. The circuit inputs $\mathbf{x}$ and $\mathbf{y}$ and output $\mathbf{r}$ were kept constant in each trial using sample and hold units.

TABLE I
CIRCUIT PARAMETERS

Type	Parameter	Value
Power source	V_{DD}	$10\,V$
NMOS	K	$5\,\mathrm{AV}^{-2}$
	V_T	$1.7\,\mathrm{V}$
PMOS	K	$5\,\mathrm{AV}^{-2}$
	V_T	$1.4\,\mathrm{V}$
Memristor	$\bar{g}$	$1\,\mu S$
	$\hat{g}$	$180\,\mu S/\,(\mathrm{V}\cdot\mathrm{s})$
Timing	T	$0.05\,\mathrm{s}$
	T_{wr}	$0.56\,T$
Scaling	a	$0.1\,V$
	b	T_{wr}
	c^{-1}	$10\,nA$

B. Basic Functionality

First, the basic operation of the proposed synaptic grid circuit is examined numerically in a toy example.

A small 2×2 synaptic grid circuit is simulated for time 10 T (10 read-write cycles) with a simple piecewise constant input

$$\mathbf{x} = (x_1, x_2) = (0.8, -0.4) \cdot \mathrm{sign}(t - 5T) \tag{39}$$

and a constant input

$$\mathbf{y} = (y_1, y_2) = (0.2, -0.1). \tag{40}$$

In Fig. 7, the resulting circuit outputs are shown, together with the memristors' voltages and conductances. The correct basic operation of the circuit is verified.

TABLE II
LEARNING PARAMETERS FOR EACH DATASET

Parameter	Task 1	Task 2
Neuronal layer sizes	$30 \to 1$	$4 \to 4 \to 3$
Training set size	284	75
Test set size	285	75
η	0.1	0.1

1) In the first read cycle, the memristors are used to generate the output (23). The voltage trace on the (n, m) memristor is a $\pm a x_m$ bipolar pulse, as expected from (16). This results in a nondestructive (a zero net change in conductance), as expected from (17).
2) In the second read cycle, the memristors are used to generate the output (35). The voltage trace on the (n, m) memristor is a $\pm a y_n$ bipolar pulse [as expected from (32)], again resulting in a nondestructive read.
3) In the write cycle, the stored weights are incremented according to (28). As expected from (24), the (n, m) memristor is subjected to a voltage pulse of amplitude $\text{sign}(y_n) a x_m$ with duration $b|y_n|$. Furthermore, there is an $\hat{g} a b x_m y_n$ increment in memristor conductance following (27).
4) In the output of the circuit is $r_n = \sum_m a c \hat{g} s_{nm} x_m$ following (5), and (18)–(23).

C. Learning Performance

The synaptic grid circuit model is used to implement a SNN and a MNN, trainable by the online gradient descent algorithm. To demonstrate that the algorithm is indeed implemented correctly by the proposed circuit, the circuit performance has been compared to an algorithmic implementation of the MNNs in (MATLAB) software. Two standard tasks are used:

1) the Wisconsin Breast Cancer diagnosis task [56] (linearly separable);
2) the Iris classification task [56] (not linearly separable).

The first task was evaluated using an SNN circuit, similarly to Fig. 1. Note this task has only a single output (yes/no), so the synaptic grid (Fig. 2) has only a single row. The second task is evaluated on a two-layer MNN circuit, similarly to Fig. 4. The learning parameters are described in Table II. Fig. 8 shows the training error (performance during training phase) of the following:

1) the algorithm;
2) the proposed circuit (implementing the algorithm);
3) the circuit, with about 10% noise and 30% variability (Section V).

In addition, Table III shows the test error—the error estimated on the test set after training was over (The standard deviation (Std) was calculated as $(P_e(1 - P_e)/N_{\text{test}})^{1/2}$, with N_{test} being the number of samples in the test set and P_e being the test error). On each of the two tasks, the training performance (i.e., on the training set) of the circuit and the algorithm similarly improves, finally reaching a similar test error. These results indicate that the proposed circuit design can precisely implement MNNs trainable with online gradient descent, as expected, from the derivations in Section III. Moreover, the circuit exhibits considerable robustness, as its performance is only mildly affected by significant levels of noise and variation.

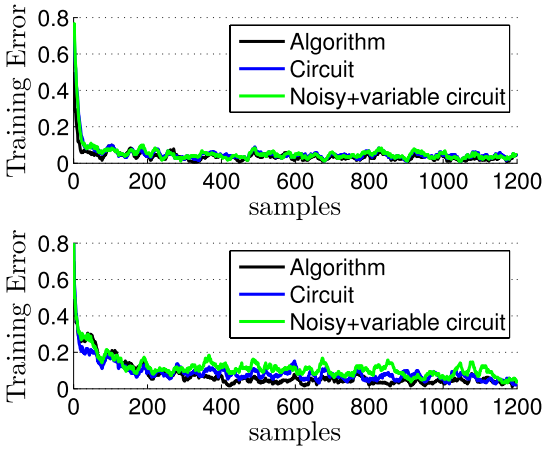

Fig. 8. Circuit evaluation—training error for two datasets. (a) Task 1—Wisconsin breast cancer diagnosis. (b) Task 2—Iris classification.

TABLE III
CIRCUIT EVALUATION—TEST ERROR (MEAN ± STD) FOR (a) ALGORITHM, (b) CIRCUIT, AND (c) CIRCUIT WITH NOISE AND VARIABILITY

	Task 1	Task 2
(a)	$1.3\% \pm 0.7\%$	$2.9\% \pm 2\%$
(b)	$1.5\% \pm 0.7\%$	$2.8\% \pm 1.9\%$
(c)	$1.5\% \pm 0.7\%$	$4.7\% \pm 2.4\%$

Implementation Details: The SNN and the MNN were implemented using the (SimElectronics) synaptic grid circuit model, which was described in Section VI-A. The detailed schematics of the two-layer MNN model are again given in [39] (as an HTML file, which may be opened using Firefox), and also in the code. All simulations are done using a desktop computer with core i7 930 processor, a Windows 8.1 operating system, and a MATLAB 2013b environment. Each circuit simulation takes about one day to complete (note the software implementation of the circuit does not exploit the parallel operation of the circuit hardware). For both tasks, we used the standard training and testing procedures [6]. For each k sample from the dataset, the attributes are converted to a M-long vector $\mathbf{x}^{(k)}$, and each label is converted to a N-long binary vector $\mathbf{d}^{(k)}$. The training set is repeatedly presented, with samples at random order. The inputs mean was subtracted, and they were normalized as recommended by [6], with additional rescaling done in dataset 1, to ensure that the inputs fall within the circuit specification 1, i.e., meet the requirement specified by (13). The performance was averaged over 10 repetitions (of training and testing) and the training error was also averaged over the previous 20 samples. The initial weights were sampled independently from a symmetric uniform distribution. For both tasks, in the output layer,

a softmax activation function was used

$$(\sigma(\mathbf{x}))_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

together with a Cross entropy error (37) since this combination is known to improve performance [57]. In task 2, in the two-layer MNN, the neuronal activation functions in the first (hidden) layer were set as

$$\sigma(x_i) = 1.7159 \tanh\left(\frac{2x_i}{3}\right)$$

as recommended in [6]. Other parameters are given in Tables I and II. In addition to the inputs from the previous layer, the commonly used bias input is implemented in the standard way (i.e., at each circuit, the neuronal input $\mathbf{x}$ is extended with an additional component equal to a constant one).

VII. Discussion

As explained in Sections II and IV, two major computational bottlenecks of MNNs and many ML algorithms, are given by a 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 and two transistors. The physical grid-like structure of the arrays implements the matrix × vector product operation (4) using analog summation of currents, while the memristor (nonlinear) dynamics enable us to perform the vector × vector outer product operation (10), using time × voltage encoding paradigm. The idea to use a resistive grid to perform matrix × vector product operation is not new (e.g., [58]). The main novelty of this paper is the use of memristors together with time × voltage encoding, which allows us to perform a mathematically accurate vector × vector outer product operation in the learning rule using a small number of components.

A. Previous CMOS-Based Designs

As mentioned in the introduction, CMOS hardware designs that specifically implement online 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., CPUs, digital signal processors, and GPUs). However, squeezing the required circuit for both the computation and the update phases (two configurable multipliers, for the matrix × vector product and vector × vector outer product, and a memory element to store the synaptic weight) 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 [67, Table I] 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 × vector product operation in the computation phase, rather than the computational bottleneck of the vector × vector outer product operation in the update phase. In addition, these solutions are only useful in cases where the training is not continuous and is done in a predeployment phase or in special reconfiguration phases during the operation. Other designs implement nonstandard (e.g., perturbation algorithms [68]) or specifically tailored learning algorithms (e.g., modified backpropagation for spiking neurons [69]). However, it remains to be seen whether such algorithms are indeed scalable.

TABLE IV
Hardware Designs of Artificial Synapses Implementing Scalable Online Learning Algorithms

Design	#Transistors	Comments
Proposed design	2 (+1 memristor)	
[59]	2	Also requires UV light + Weights decay ~ minutes
[60]	6	Weights only increase (unusable)
[61, 62]	39	Must keep training
[63]	52	Must keep training
[64]	92	Weights decay ~ hours
[65]	83	Also requires a "weight unit"
[66]	150	

Hardware designs of artificial synaptic arrays that are capable of implementing common (scalable) online gradient-descent-based learning, are listed in Table IV. For large arrays (i.e., large M and N), the effective transistor count per synapse (where resistors and capacitors were also counted as transistors) is approximately proportional to the required area and average static power usage of the circuit.

The smallest synaptic circuit [59] 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 [60] 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 [62], [70] suggested a grid design using the CMOS Gilbert multipliers, resulting in 39 transistors per synapse. In a similar design [63], 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. Therefore, they require constant retraining (practically acting as refresh). Such retraining is required also in each startup or, alternatively, reading out the weights into an auxiliary memory—a solution that requires a mechanism for reading out the synaptic weights. The larger design in [64] (92 transistors) also has weight decay, but with a slow hours-long timescale. The device in [65] (83 transistors and an unspecified weight unit that stores the weights) does not report to have weight decay, apparently since digital storage is used. This is also true for [66] (150 transistors).

B. Memristor-Based Designs: Expected Benefits and Technical Issues

The proposed memristor-based design should resolve the main obstacles of the above CMOS-based designs, and provide a compact nonvolatile [71] circuit. Area and power consumption are expected to be reduced by a factor of 13–50, in comparison with 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 the CMOS-based designs is the fact that although most of these designs are two decades old, they have not been incorporated into commercial products. It is fair only 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. However, a physical implementation of a memristor-based neural network should be feasible, as was demonstrated in a recent work [31]—where a different memristor-based design was manufactured and tested.

The hardware design in [31] of a SNN with binary outputs demonstrated a successful online training using the popular perceptron algorithm. The current proposed design allows more flexibility, since it can be used to train general MNNs (considered to be much more powerful than SNNs [33]) using the scalable online gradient descent algorithm (backpropagation). In addition, the current proposed design can be also used to implement the perceptron algorithm (used in [31]), since it is very similar to the Adaline algorithm (Fig. 1). Due to this similarity, both designs should encounter similar technical issues in a concrete implementation.

Encouragingly, the memristor-based neural network circuit in [31] is able to achieve good performance, despite of the following issues: 1) noisy memristor dynamics affect the accuracy of the weight update; 2) variations in memristor parameters generate similar variations in the learning rates (here η); and 3) the nonlinearity of the conductivity [here $G(s(t))$] can have a saturating effect on the weights (resulting in bounded weights). Overcoming issues 1) and 2) suggests that training MNNs should be relatively robust to noise and variations, as argued here (Fig. 5) and demonstrated numerically (Fig. 8 and Table III). Overcoming issue (3) suggests that the saturating effect of the nonlinearity $G(s(t))$ on the weights is not catastrophic. This could be related again to the robustness of gradient descent (Fig. 5). Moreover, bounding the weights magnitude can be desirable and various regularization methods are commonly used to achieve this effect in MNNs. More specifically, a saturating nonlinearity on the weights can even improve performance [72]. Other types of nonlinearity may also be beneficial [73].

C. Circuit Modifications and Generalizations

The specific parameters that were used for the circuit evaluation (Table I) are not strictly necessary for the proper execution of the proposed design and are only used to demonstrate its applicability. For example, it is straightforward to show that K, V_T, and $\bar{g}$ have little effect [as long as (13) and (14) hold], and different values of $\hat{g}$ can be adjusted for by rescaling the constant c, which appears in (20) and (34). 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 [74] to milliseconds [55]. Here, the parameters were taken from a millisecond-timescale memristor [55]. The actual speed of the proposed circuit would critically depend on the timescales of commercially available memristor devices.

Additional important modifications of the proposed circuit are straightforward. In [Appendix A, 39], it is explained how to modify the circuit to work with more realistic memristive devices [40, 14], instead of the classical memristor model [11], given a few conditions. In [Appendix C, 39], 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. 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. In addition, it is straightforward to replace the simple outer product update rule in (10) by more general update rules of the form

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

where f_i, g_j are some functions. Finally, it is possible to adaptively modify the learning rate η during training [e.g., by modifying α in (36)].

VIII. Conclusion

A novel method to implement scalable online gradient descent learning in multilayer neural networks through local update rules is proposed based on the emerging memristor technology. The proposed method is based on an artificial synapse using one memristor to store the synaptic weight and two CMOS transistors to control the circuit. The correctness of the proposed synapse structure exhibits a similar accuracy to its equivalent software implementation, while the proposed structure shows high robustness and immunity to noise and parameter variability.

Such circuits may be used to implement large-scale online learning in multilayer neural networks, as well as other learning systems. 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, operating with low static power and good robustness. Such brain-like properties may give a significant boost to the field of neural networks and learning systems.

Acknowledgment

The authors would like to thank E. Friedman, I. Hubara, R. Meir, and U. Weiser for their support and helpful comments, and E. Rosenthal and S. Greshnikov for their contribution to the SPICE simulations.

References

[1] Q. V. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. S. Corrado, J. Dean, and A. Y. Ng, "Building high-level features using large scale unsupervised learning," in *Proc. ICML*, Edinburgh, Scotland, Jun. 2012, pp. 81–88.

[2] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proc. CVPR*, Providence, RI, USA, Jun. 2012, pp. 3642–3649.

[3] J. Dean *et al.*, "Large scale distributed deep networks," in *Proc. NIPS*, Lake Tahoe, NV, USA, Dec. 2012, pp. 1–9.

[4] R. D. Hof, "Deep learning," *MIT Technol. Rev.*, Apr. 2013.

[5] D. Hernandez, "Now you can build Google's $1M artificial brain on the cheap," *Wired*, Jun. 2013, pp. 9–48.

[6] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural Networks: Tricks of the Trade*, G. Montavon, G. B. Orr, and K.-R. Müller, Eds., 2nd ed. Heidelberg, Germany: Springer-Verlag, 2012.

[7] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomputing*, vol. 74, nos. 1–3, pp. 239–255, Dec. 2010.

[8] A. R. Omondi, "Neurocomputers: A dead end?" *Int. J. Neural Syst.*, vol. 10, no. 6, pp. 475–481, 2000.

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

[10] M. M. Waldrop, "Neuroelectronics: Smart connections," *Nature*, vol. 503, no. 7474, pp. 22–24, Nov. 2013.

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

[12] 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, Mar. 2008.

[13] F. Corinto and A. Ascoli, "A boundary condition-based approach to the modeling of memristor nanostructures," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 59, no. 11, pp. 2713–2726, Nov. 2012.

[14] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "TEAM: ThrEshold adaptive memristor model," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 60, no. 1, pp. 211–221, Jan. 2013.

[15] S. Kvatinsky, Y. H. Nacson, Y. Etsion, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based multithreading," *Comput. Archit. Lett.*, vol. 13, no. 1, pp. 41–44, Jul. 2014.

[16] Z. Guo, J. Wang, and Z. Yan, "Passivity and passification of memristor-based recurrent neural networks with time-varying delays," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 11, pp. 2099–2109, Nov. 2014.

[17] L. Wang, Y. Shen, Q. Yin, and G. Zhang, "Adaptive synchronization of memristor-based neural networks with time-varying delays," *IEEE Trans. Neural Netw. Learn. Syst.*, to be published.

[18] G. Zhang and Y. Shen, "Exponential synchronization of delayed memristor-based chaotic neural networks via periodically intermittent control," *Neural Netw.*, vol. 55, pp. 1–10, Jul. 2014.

[19] D. Querlioz, O. Bichler, and C. Gamrat, "Simulation of a memristor-based spiking neural network immune to device variations," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, San Jose, CA, USA, Jul./Aug. 2011, pp. 1775–1781.

[20] 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 Neurosci.*, vol. 5, p. 26, Jan. 2011.

[21] O. Kavehei *et al.*, "Memristor-based synaptic networks and logical operations using in-situ computing," in *Proc. 7th Int. Conf. Intell. Sensors, Sensor Netw. Inf. Process.*, Adelaide, SA, Australia, Dec. 2011, pp. 137–142.

[22] 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.

[23] Y. Kim, Y. Zhang, and P. Li, "A digital neuromorphic VLSI architecture with memristor crossbar synaptic array for machine learning," in *Proc. IEEE Int. SOC Conf. (SOCC)*, Niagara Falls, NY, USA, Sep. 2012, pp. 328–333.

[24] W. Chan and J. Lohn, "Spike timing dependent plasticity with memristive synapse in neuromorphic systems," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Brisbane, QLD, Australia, Jun. 2012, pp. 1–6.

[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 Neurosci.*, vol. 7, p. 2, Jan. 2013.

[26] D. Querlioz, O. Bichler, P. Dollfus, and C. Gamrat, "Immunity to device variations in a spiking neural network with memristive nanodevices," *IEEE Trans. Nanotechnol.*, vol. 12, no. 3, pp. 288–295, May 2013.

[27] R. Legenstein, C. Naeger, and W. Maass, "What can a neuron learn with spike-timing-dependent plasticity?" *Neural Comput.*, vol. 17, no. 11, pp. 2337–2382, Mar. 2005.

[28] D. Chabi, W. Zhao, D. Querlioz, and J.-O. Klein, "Robust neural logic block (NLB) based on memristor crossbar array," in *Proc. IEEE/ACM Int. Symp. IEEE Nanosc. Archit. (NANOARCH)*, San Diego, CA, USA, Jun. 2011, pp. 137–143.

[29] H. Manem, J. Rajendran, and G. S. Rose, "Stochastic gradient descent inspired training technique for a CMOS/nano memristive trainable threshold gate array," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 59, no. 5, pp. 1051–1060, May 2012.

[30] M. Soltiz, D. Kudithipudi, C. Merkel, G. S. Rose, and R. E. Pino, "Memristor-based neural logic blocks for nonlinearly separable functions," *IEEE Trans. Comput.*, vol. 62, no. 8, pp. 1597–1606, Aug. 2013.

[31] F. Alibart, E. Zamanidoost, and D. B. Strukov, "Pattern classification by memristive crossbar circuits using *ex situ* and *in situ* training," *Nature Commun.*, vol. 4, p. 2072, May 2013.

[32] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychol. Rev.*, vol. 65, no. 6, pp. 386–408, Nov. 1958.

[33] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control, Signals, Syst.*, vol. 2, no. 4, pp. 303–314, 1989.

[34] L. Bottou and O. Bousquet, "The tradeoffs of large scale learning," in *Optimization for Machine Learning*, S. Sra, S. Nowozin, and S. J. Wright, Eds. Cambridge, MA, USA: MIT Press, 2011, p. 351.

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

[36] S. P. Adhikari, C. Yang, H. Kim, and L. O. Chua, "Memristor bridge synapse-based neural network and its learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 9, pp. 1426–1435, Sep. 2012.

[37] R. Hasan and T. M. Taha, "Enabling back propagation training of memristor crossbar neuromorphic processors," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Beijing, China, Jul. 2014, pp. 21–28.

[38] Z. Vasilkoski *et al.*, "Review of stability properties of neural plasticity rules for implementation on memristive neuromorphic hardware," in *Proc. Int. Joint Conf. Neural Netw.*, San Jose, CA, USA, Jul./Aug. 2011, pp. 2563–2569.

[39] *The Supplementary Material*. [Online]. Available: http://ieeexplore.ieee.org.

[40] L. O. Chua and S. M. Kang, "Memristive devices and systems," *Proc. IEEE*, vol. 64, no. 2, pp. 209–223, Feb. 1976.

[41] M. D. Pickett *et al.*, "Switching dynamics in titanium dioxide memristive devices," *J. Appl. Phys.*, vol. 106, no. 7, p. 074508, 2009.

[42] J. Strachan *et al.*, "State dynamics and modeling of tantalum oxide memristors," *IEEE Trans. Electron Devices*, vol. 60, no. 7, pp. 2194–2202, Jul. 2013.

[43] B. Widrow and M. E. Hoff, "Adaptive switching circuits," Stanford Electron. Labs, Stanford Univ., Stanford, CA, USA, Tech. Rep., 1960.

[44] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1985.

[45] E. Oja, "Simplified neuron model as a principal component analyzer," *J. Math. Biol.*, vol. 15, no. 3, pp. 267–273, Nov. 1982.

[46] C. M. Bishop, *Pattern Recognition and Machine Learning*. Singapore: Springer-Verlag, 2006.

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

[48] S. Kvatinsky, N. Wald, E. Satat, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based material implication (IMPLY) logic: Design principles and methodologies," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 10, pp. 2054–2066, Oct. 2014.

[49] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, U.K.: Oxford Univ. Press, 1995.

[50] R. Sarpeshkar, "Analog versus digital: Extrapolating from electronics to neurobiology," *Neural Comput.*, vol. 10, no. 7, pp. 1601–1638, Oct. 1998.

[51] *The Mosis Service*. [Online]. Available: http://www.mosis.com, accessed Nov. 21, 2012.

[52] 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 *Proc. IEEE 57th Electron. Compon. Technol. Conf. (ECTC)*, Sparks, NV, USA, May/Jun. 2007, pp. 1659–1666.

[53] M. Hu, H. Li, Y. Chen, X. Wang, and R. E. Pino, "Geometry variations analysis of TiO_2 thin-film and spintronic memristors," in *Proc. 16th Asia South Pacific Design Autom. Conf.*, Yokohama, Japan, Jan. 2011, pp. 25–30.

[54] *SimElectronics*. [Online]. Available: http://www.mathworks.com/products/simelectronics/, accessed Nov. 21, 2012.

[55] 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," *Appl. Phys. A*, vol. 102, no. 4, pp. 857–863, Feb. 2011.
[56] K. Bache and M. Lichman. (2013). *UCI Machine Learning Repository*. [Online]. Available: http://archive.ics.uci.edu/ml
[57] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *Proc. 7th Int. Conf. Document Anal. Recognit.*, vol. 1. Edinburgh, Scotland, Aug. 2003, pp. 958–963.
[58] D. B. Strukov and K. K. Likharev, "Reconfigurable nano-crossbar architectures," in *Nanoelectronics and Information Technology*, R. Waser, Ed. New York, NY, USA: Wiley, 2012, pp. 543–562.
[59] G. Cauwenberghs, C. F. Neugebauer, and A. Yariv, "Analysis and verification of an analog VLSI incremental outer-product learning system," *IEEE Trans. Neural Netw.*, vol. 3, no. 3, pp. 488–497, May 1992.
[60] H. C. Card, C. R. Schneider, and W. R. Moore, "Hebbian plasticity in MOS synapses," *IEE Proc. F, Radar Signal Process.*, vol. 138, no. 1, pp. 13–16, Feb. 1991.
[61] C. Schneider and H. Card, "Analogue CMOS Hebbian synapses," *Electron. Lett.*, vol. 27, no. 9, pp. 785–786, Apr. 1991.
[62] H. C. Card, C. R. Schneider, and R. S. Schneider, "Learning capacitive weights in analog CMOS neural networks," *J. VLSI Signal Process. Syst. Signal, Image Video Technol.*, vol. 8, no. 3, pp. 209–225, Oct. 1994.
[63] M. Valle, D. D. Caviglia, and G. M. Bisio, "An experimental analog VLSI neural network with on-chip back-propagation learning," *Analog Integr. Circuits Signal Process.*, vol. 9, no. 3, pp. 231–245, Apr. 1996.
[64] T. Morie and Y. Amemiya, "An all-analog expandable neural network LSI with on-chip backpropagation learning," *IEEE J. Solid-State Circuits,* vol. 29, no. 9, pp. 1086–1093, Sep. 1994.
[65] C. Lu, B.-X. Shi, and L. Chen, "An on-chip BP learning neural network with ideal neuron characteristics and learning rate adaptation," *Analog Integr. Circuits Signal Process.*, vol. 31, no. 1, pp. 55–62, Apr. 2002.
[66] T. Shima, T. Kimura, Y. Kamatani, T. Itakura, Y. Fujita, and T. Iida, "Neuro chips with on-chip back-propagation and/or Hebbian learning," *IEEE J. Solid-State Circuits*, vol. 27, no. 12, pp. 1868–1876, Dec. 1992.
[67] C. S. Lindsey and T. Lindblad, "Survey of neural network hardware," *Proc. SPIE*, vol. 2492, pp. 1194–1205, Apr. 1995.
[68] G. Cauwenberghs, "A learning analog neural network chip with continuous-time recurrent dynamics," in *Proc. NIPS*, Golden, CO, USA, Nov. 1994, pp. 858–865.
[69] H. Eguchi, T. Furuta, H. Horiguchi, S. Oteki, and T. Kitaguchi, "Neural network LSI chip with on-chip learning," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Seattle, WA, USA, Jul. 1991, pp. 453–456.
[70] C. Schneider and H. Card, "CMOS implementation of analog Hebbian synaptic learning circuits," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, vol. i. Seattle, WA, USA, vol. 1, Jul. 1991, pp. 437–442.
[71] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, no. 7, p. 075201, Feb. 2012.
[72] D. Soudry, I. Hubara, and R. Meir, "Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights," in *Proc. NIPS*, Montreal, QC, Canada, Dec. 2014, pp. 963–971.
[73] M. Milev and M. Hristov, "Analog implementation of ANN with inherent quadratic nonlinearity of the synapses," *IEEE Trans. Neural Netw.*, vol. 14, no. 5, pp. 1187–1200, Sep. 2003.
[74] 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.

Daniel Soudry received the B.Sc. degree in electrical engineering and physics and the Ph.D. degree in electrical engineering from the Technion-Israel Institute of Technology, Haifa, Israel, in 2008 and 2013, respectively.

He is currently a Gruss Lipper Post-Doctoral Fellow with the Department of Statistics, Center of Theoretical Neuroscience, and the Grossman Center for the Statistics of Mind, Columbia University, New York, NY, USA. His current research interests include modeling the nervous system and its components, Bayesian methods for neural data analysis and inference in neural networks, and hardware implementation of neural systems.

Dotan Di Castro received the B.Sc., M.Sc., and Ph.D. degrees from the Technion-Israel Institute of Technology, Haifa, Israel, in 2003, 2006, and 2010, respectively.

He was with IBM Research Labs, Haifa, from 2000 to 2004, He was involved in several startup companies from 2009 to 2013. He is currently with Yahoo! Labs, Haifa, where he is investigating information processing in very large scale systems. His current research interests include machine learning (in particular, reinforcement learning), computer vision, and large-scale hierarchical learning systems.

Asaf Gal received the B.Sc. degree in physics and electrical engineering from the Technion-Israel Institute of Technology, Haifa, Israel, in 2004, and the Ph.D. degree in computational neuroscience from the Hebrew University of Jerusalem, Jerusalem, Israel, in 2013.

He is currently a Clore Post-Doctoral Fellow with the Department of Physics of Complex Systems, Weizmann Institute of Science, Rehovot 76100, Israel. His current research interests include theoretically oriented study of biological systems, biophysics, and the application of complex systems science to experiments in biology.

Avinoam Kolodny received the Ph.D. degree in microelectronics from the Technion-Israel Institute of Technology (Technion), Haifa, Israel, in 1980.

He joined Intel Corporation, Santa Clara, CA, USA, where he was involved in research and development in the areas of device physics, very large scale integration (VLSI) circuits, electronic design automation, and organizational development. He has been a member of the Faculty of Electrical Engineering with Technion since 2000. His current research interests include interconnects in VLSI systems, at both physical and architectural levels.

Shahar Kvatinsky received the B.Sc. degree in computer engineering and applied physics and the M.B.A. degree from the Hebrew University of Jerusalem, Jerusalem, Israel, in 2009 and 2010, respectively, and the Ph.D. degree in electrical engineering from the Technion-Israel Institute of Technology, Haifa, Israel, in 2014.

He was with Intel Corporation, Santa Clara, CA, USA, as a Circuit Designer, from 2006 to 2009. He is currently a Post-Doctoral Research Fellow with Stanford University, Stanford, CA, USA. His current research interests include circuits and architectures with emerging memory technologies and design of energy efficient architectures.