Learning by mistakes in memristor networks

Juan Pablo Carbajal
Institute for Energy Technology, University of Applied Sciences of Eastern Switzerland, Oberseestrasse 10, 8640 Rapperswil, Switzerland

Daniel A. Martin and Dante R. Chialvo*
Center for Complex Systems and Brain Sciences (CEMSC^3) and Instituto de Ciencias Físicas, CONICET, Escuela de Ciencia y Tecnología, Universidad Nacional de General San Martín, Campus Miguelete, CP 1650, 25 de Mayo y Francia, San Martín, Buenos Aires, Argentina

(Received 25 January 2022; accepted 20 April 2022; published 11 May 2022)

Recent results revived the interest in the implementation of analog devices able to perform brainlike operations. Here we introduce a training algorithm for a memristor network which is inspired by previous work on biological learning. Robust results are obtained from computer simulations of a network of voltage-controlled memristive devices. Its implementation in hardware is straightforward, being scalable and requiring very little peripheral computation overhead.

DOI: 10.1103/PhysRevE.105.054306

I. INTRODUCTION

In the past decade we have witnessed an explosion in the interest in neuromorphing, i.e., adaptive devices inspired by brain principles. Many of the current efforts focus on the replication of the dynamics of a single neuron, using a diversity of technologies including magnetics, optics, atomic switches, etc. [1]. While the emulation of a single neuron seems achievable with the present technology, we still lack learning algorithms to train large interconnected neuronlike elements without resorting to peripheral computation overhead.

In thinking about this issue, it is soon realized that *in vivo* biological learning exhibits important features which are not presently considered in neuromorphing implementations. The most relevant one, in the context of our work, is the fact that the only information a biological neuron has at its disposal to modify its synapses is either global or local. In other words, in real brains, there is no peripheral computation overhead; the strength of the synaptic weight between any two given neurons is a function of the activity of its immediate neighbors and/or (through some so-called neuromodulators) some global state (or partial region) of the brain, resulting from success or frustration in achieving some goal (or being happy, angry, excited, sleepy, etc.). These observations have led to the proposal [2–7] of a simple neural network model able to learn simple input-output associations.

The present article instantiates a translation of the work of Refs. [2,3] into the realm of memristive networks. The main objective is to design a device working on the principles described therein, able to be fully implemented in hardware, requiring no access to the inner structure of the network and minimal (i.e., one ammeter, one switch, and two batteries) external processing.

This article is organized as follows. First we will review previous work [2–7] describing a self-organized process by which biological learning may proceed. Such work is the inspiration for the algorithm proposed here to train a network of memristors [8,9], which is introduced after that. Subsequently, the main results describing the simulation results obtained from a three-layer feedforward network are presented. The paper closes with a short list of expected hurdles to surpass and other possible similar implementations. Numerical details are described in the Appendix, together with some miscellaneous observations.

II. ALGORITHM, MODEL, AND OBSERVABLES

A. Toy model of biological learning

Two decades ago, Chialvo and Bak [3] introduced an unconventional model of learning which emphasized self-organization. In that work they reexamined the commonly held view that learning and memory necessarily require potentiation of synapses. Instead, they suggested that, for a naive neuronal network, the process of learning involves making more mistakes than successful choices; thus the process of adapting the synapses would have more opportunities to punish the mistakes than to positively reinforce the successes. Consequently, their learning strategy used two steps: The first involves extremal dynamics to determine the propagation of the activity through the nodes and the second using synaptic depression to decrease the weights involved in the undesired (i.e., mistaken) outputs. The first step implies the selection of only the strongest synapses for propagating the activity. The second step assumes that active synaptic connections are temporarily tagged and subsequently depressed if the resulting output turns out to be unsuccessful. Thus, all the synaptic adaptation leading to learning is driven only by the mistakes.

The toy model considered an arbitrary network of nodes connected by weights. Although almost any network topology

*dchialvo@gmail.com

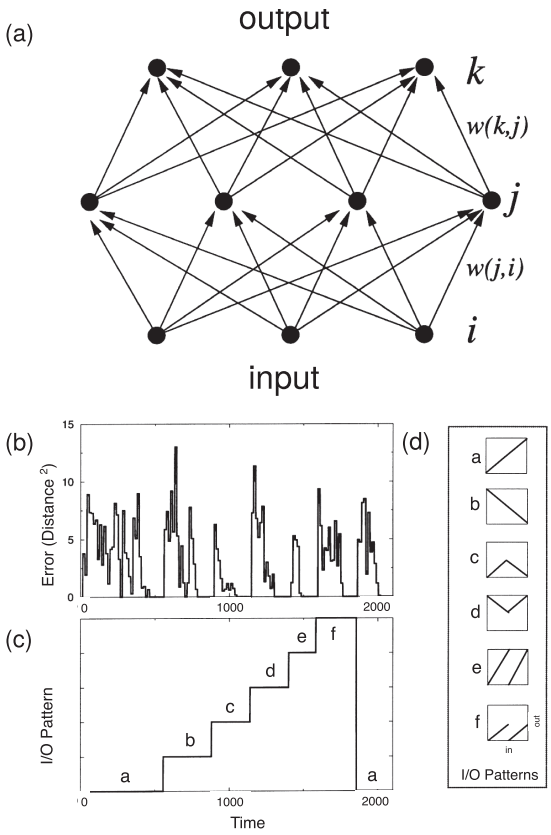

FIG. 1. (a) Example of a three-layer network with three input nodes, four intermediary nodes, and three output nodes. Each input node has a synaptic connection to every intermediary node, and each intermediary node has a similar connection to every output node. (b)–(d) Typical run to learn the six simple input-output patterns (i.e., maps) for a network of six input neurons, 300 middle neurons, and six output neurons. As noted in (b), the error eventually reaches zero, after which learning of a new pattern (c) is attempted. The list of maps is shown in (d). (Figure has been redrawn from [3].)

can be used, for the sake of description we discuss the simplest version of a three-layer feedforward network [see Fig. 1(a)]. To describe the working principle, let us suppose that we wish to train the network to learn an arbitrary map. To be precise, a map is an association of each input neuron with an output neuron. For instance, the identity map is one where each input neuron is associated with the corresponding output neuron (the first input neuron is associated with the first output neuron, the second input neuron is associated with the second output neuron, and the same relation holds for all other input neurons) and a random map is one where one output neuron is chosen randomly for each input neuron.

The learning algorithm needs to modify the network's weights in such a way that a given input neuron connects to the prescribed output neuron. The entire dynamical process goes as follows.

(1) Activate an input neuron i chosen randomly from the set established by the task to learn.

(2) Activate the neuron j_m in the middle layer connected with the input neuron i with the largest $w(j, i)$.

(3) Activate the output neuron k_m with the largest $w(k, j_m)$.

(4) If the output k happens to be the desired one, nothing is done.

(5) Otherwise, that is, if the output is not correct, $w(k_m, j_m)$ and $w(j_m, i)$ are both reduced (depressed) by an amount δ.

(6) Go back to step 1. Another input neuron from the task set is randomly chosen and the process is repeated.

The process involves a principle of extremal dynamics (here simplified by choosing the strongest weights) followed, in the case of incorrect output, by negative feedback (i.e., step 5 of the adaptation). The only parameter of the model is δ, but, at least in the numerical simulations, it is not crucial at all, because its only role is to prevent the same path from the input to the undesired output to be selected more than once. Numerical explorations with this simple model showed that step 5 above can be modified in many different ways (including choosing random values) without serious consequences, as long as it makes less probable the persistence of "wrong paths." Initial values of $w(i, j)$ and $w(j, k)$ are not relevant either [3].

Note that the toy model omits consideration of the neuronal dynamics: It is not necessary to introduce spikes whose only role would be to propagate activity across the network. Since propagation occurs most often (statistically speaking) across the strongest synapses, the toy model omits including spikes and directly selects the strongest paths, as done in steps 2 and 3 of the algorithm.

In Figs. 1(b) and 1(c) we reproduce the results of a typical simulation in which a few simple maps [indicated in Fig. 1(d)] are successively learned by the model (see details in [3]). The error in learning map a, computed as the squared distance between the actual output and the desired one, is seen to fluctuate until eventually vanishing at time $\simeq$600 [see Fig. 1(b)]. After that, the network is given the task to learn map b (which is achieved at time $\simeq$800), map c, and so on. Interference between maps is expected for a relatively small system size, since the same path can be chosen by chance for two different input-output maps. As was discussed earlier in Refs. [3,4], a system trained under these premises is robust with respect to noise, in the sense that depression of synaptic weights will self-adjust proceeding to correct the errors, until eventually achieving the desired outputs. Another interesting property of this setup is that the learning time goes down with the size of the middle layer, a fact that is easily understood since the learning process implies finding and keeping the strongest paths between the input and the desired nodes in the output layer. This and other scaling relations can be found in Refs. [3–7].

B. Memristor model

Now we turn to discuss how to implement the toy model just described on a network of memristors. Memristive devices are a family of two-terminal devices whose resistance evolves according to the bias and currents they experience [10]. In analogy to long-term potentiation and depression taking place in neuronal synapses, memristor resistances can be increased or decreased through the application of relatively high voltage differences or currents.

In this article we consider voltage-controlled memristors with a threshold [11], whose resistance R can take any value between $R_{\min}$ and $R_{\max}$. When a voltage difference V is

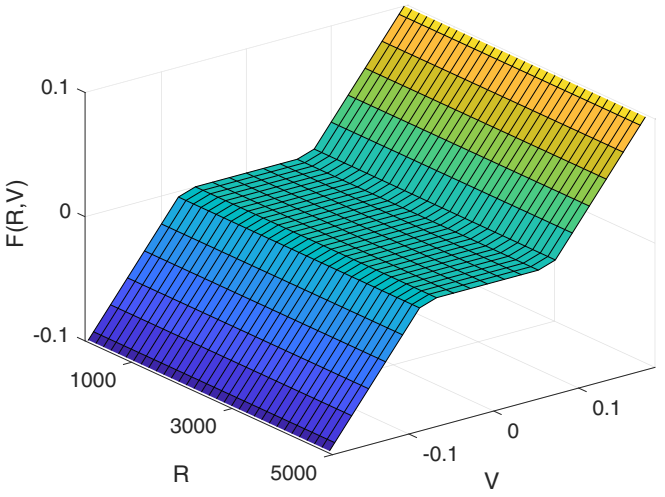

FIG. 2. Memristor behavior as a function of resistance and applied voltage. The resistance of the memristor does not change unless the absolute value of the applied voltage is greater than $V_{\updownarrow}$. Here we used $V_{\updownarrow} = 0.075$, $R_{\min} = 75$, $R_{\max} = 5000$, and $\beta = 0.9$

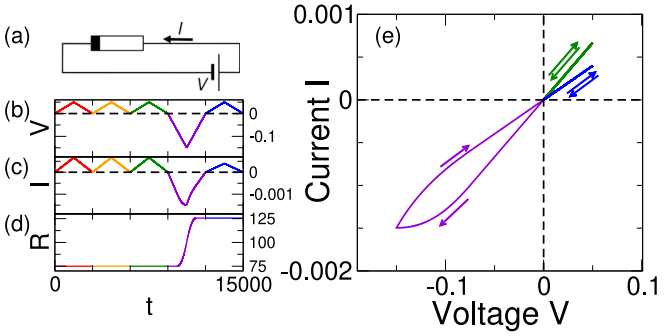

FIG. 3. Behavior of a voltage-controlled memristor connected to the circuit shown in (a). (b) Typical changes in the properties of the memristor as a function of a time-dependent voltage V. Also shown are (c) the resulting current I and (d) the instantaneous resistance R. (e) The same data are presented as an I-V curve. Notice that relatively small voltage excursions (i.e., upward triangular sweeps) do not change the device resistance, while relatively large voltage excursions do, resulting in the typical hysteresis loop. The results in (b)–(e) have been colored to aid the interpretation.

applied, a current I passes through the memristor and the value of R may change, depending on V and R values.

The memristor equations can be written as

$$I = \frac{V}{R}, \tag{1}$$

$$\frac{\partial R}{\partial t} = -F(R, V), \tag{2}$$

where the function F describes the behavior of the memristor. The individual memristor dynamics is fully described by previous equations plus a definition of the function F, which describes how the characteristics of the memristor change upon applied voltage differences. For that we use the bipolar memristive system with threshold (BMS), which is described in detail in Sec. 3.2 of [11] (a different memristor model, known as the boundary condition memristor [12,13], is considered in the Appendix). According to that reference, we write

$$F(R, V) = \begin{cases} \beta(V + V_{\updownarrow}) & \text{if } V < -V_{\updownarrow},\ R < R_{\max} \\ 0 & \text{if } |V| < V_{\updownarrow} \\ \beta(V - V_{\updownarrow}) & \text{if } V > V_{\updownarrow},\ R > R_{\min}, \end{cases} \tag{3}$$

where $\beta > 0$ is the rate at which the resistance increases or decreases when a large enough voltage difference is applied. The function $F(R, V)$ is illustrated in Fig. 2. From Eqs. (1)–(3) we find that a relatively large positive voltage difference tends to decrease resistance, while a negative voltage difference tends to increase resistance on the memristor (R is not modified if the absolute value of the voltage does not exceed the threshold $V_{\updownarrow}$).

In Fig. 3 we show an example of the typical changes exhibited by the voltage-controlled memristor when subjected to voltage sources of different amplitudes. The circuit and the sign convention are depicted in Fig. 3(a), while the voltage, the current across the device, and the memristor resistance are shown in Figs. 3(b)–3(d). The voltage source applies three triangle-shaped low positive voltage pulses, which do not change the memristor's resistance, followed by a high negative voltage excursion, which results in an increase of the memristor's resistance. The final triangular low voltage pulse shows that the resulting resistance increase is permanent. This property will be used here to modify the network input-output paths, as explained in the following paragraphs.[1]

C. Memristor network

The results presented here correspond to numerical simulations of a three-layer network of memristors [14], with N_{in} input nodes, N_{bulk} bulk nodes, and N_{out} output nodes. Pairs of nodes from successive layers are connected through the BMS of Ref. [11] (see also Ref. [15]). Note that this three-layer network is equivalent to a memristor crossbar array [16] with N_{in} input nodes and N_{bulk} output nodes, connected to a second memristor crossbar array of N_{bulk} inputs and N_{out} outputs. The training algorithm uses an ammeter and a voltage source with two possible values: V_{read} (read voltage) and V_{write} (punishment or correction voltage). Memristor polarity is set in such a way that a negative V_{write} tends to increase their resistance [see Figs. 3(a) and 4(b)]. The network and control resources are as sketched in Fig. 4.

We consider a relatively small (quenched) variability in the parameters of the device: The parameters for each memristor are randomly chosen from a uniform distribution with $0.8 < \beta < 1, 0.05 < V_{\updownarrow} < 0.1, 50 < R_{\min} < 100$, and $R_{\max} = 5000$. The initial condition is set to $R = R_{\min}$. The reading step lasts one time step using a voltage value of $V_{\text{read}} = 0.0001$. The

[1]In the following, it is customary to express all resistances in terms of an arbitrary unit resistance R_0, thus omitting units. Similarly, all voltages are written in terms of a unit voltage V_0 and time is also measured in terms of time unit t_0. Consequently, currents are measured in units of V_0/R_0 and the parameter β [which describes the rate at which the memristor's resistance change (see the Appendix)] in units of $R_0/V_0/t_0$.

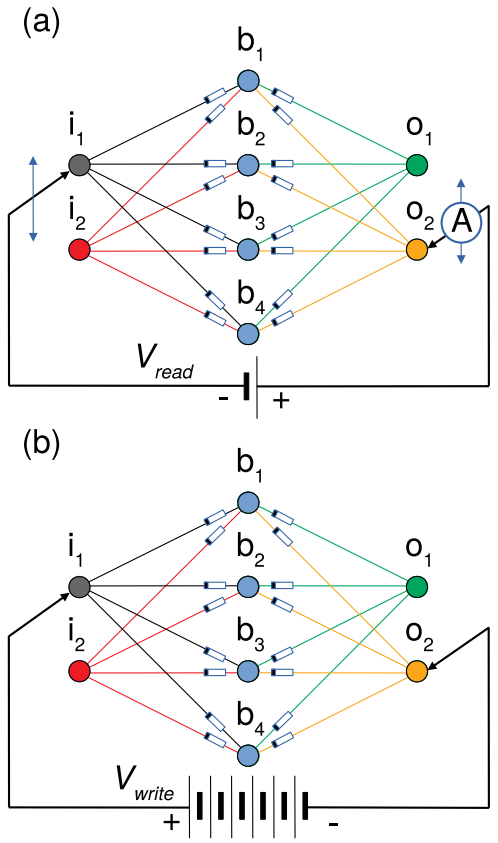

FIG. 4. Sketch of the learning algorithm for a network with $N_{\text{in}} = 2$, $N_{\text{bulk}} = 4$, and $N_{\text{out}} = 2$. In the reading step (a) a relatively small V_{read} voltage is applied and the current at each output node is measured by the ammeter. The output node with the largest current is defined as the output. If that output is not the desired one, in the correction step (b) a relatively large V_{write} voltage is applied to alter the resistance of the memristor path. The cycle is repeated until the desired map is learned.

correction step lasts five time steps using a voltage of $V_{\text{write}} = -0.2$.

Voltage values are chosen such that $|V_{\text{read}}| \ll V_{\updownarrow} < |V_{\text{write}}/2|$. In this way the memristor properties do not change in the reading step and only a few memristors change their resistance during the correction step of the algorithm.

D. Learning algorithm

Here we discuss the implementation defining a simple input-output association task, similar to the one already discussed in the preceding section for the case of the toy model of Fig. 1: For each input node i_n, we ask the memristor network to learn a randomly chosen output node.

The proposed training involves the following sequence at each training step.

(1) Randomly choose an input node i_n.

(2) Read the current flowing through all the output nodes. Here this is done by setting the i_n voltage to V_{read} and moving the ammeter tip (which closes the circuit) to each output node sequentially (o_1, o_2, etc.) while measuring the current [see Fig. 4(a)].

(3) Determine the output node with the maximum current.

(a) If the node with maximum current is the desired one, do nothing.

(b) Otherwise, that is, if the output maximum current is not at the desired node, apply V_{write} [see Fig. 4(b)] and go back to point 2.

(4) Go back to point 1.

The value of V_{read} needs to be small (such that it does not change the values of the resistances in the network); V_{write} is large and with inverted polarity, hence inducing an increase of the resistances of the network. This is the only crucial factor to ensure that the reading (in step 2) is not modifying the network conductances and conversely that the correction (in step 3b) decreases the likelihood of having large currents in the undesired paths. It is evident that the memristor learning algorithm preserves the same spirit of the earlier work: to punish wrong paths by increasing the resistance of the involved memristors. We applied step 3b a maximum of $n_{\max} = 80$ times in each training step.[2]

To collect the statistics presented here, at the end of each training step we calculate the learning error as follows. For each input node, we find the largest output (we apply V_{read} among that input node and all output nodes sequentially and take the node through which current flow is largest). We define the error as the Hamming distance from the vector of largest outputs and the desired map. If the error is null, the network has learned. Otherwise, a new training step is performed. In some cases, after the network has learned, we will consider training it with a different map. The pseudocode for this algorithm is shown in the Appendix.

III. RESULTS

Now we proceed to describe the parametric behavior of the algorithm just explained in the previous paragraphs. First we explore the dependence of the learning time on the size of the middle layer N_{bulk} for random maps. As discussed, larger values of N_{bulk} in the neuron network model of Refs. [3,4] provide more paths to the correct output, which leads to shorter learning time. We find a very similar performance for the memristive network, as shown in the results of Fig. 5, where success (the fraction of networks that have learned a map) is shown as a function of the training step and N_{bulk} for a three-layer network with $N_{\text{in}} = N_{\text{out}} = 3$, $N_{\text{in}} = N_{\text{out}} = 4$, and several values of N_{bulk}. In Figs. 5(a) and 5(b) the success as a function of the step number improves with larger N_{bulk}. This is also apparent when we plot the fraction of networks that have learned at (or before) correction step 1000 [see Figs. 5(c) and 5(d)], showing that performance is an increasing function of the middle layer size N_{bulk}. It can also be noticed, from Figs. 5(c) and 5(d), that success in equal to 1 for long enough N_{bulk}, which means that any map can be learned on those networks. Similar numerical simulation results, for a different

[2]The actual value of $n_{\max}$ does not determine whether the network learns. It only changes the number of required learning steps. Taking $n_{\max} = 1$ would make our computational implementation spend too much time reading for each correction, while setting $n_{\max} = \infty$ may make some numerical simulations get struck (i.e., in networks that do not learn).

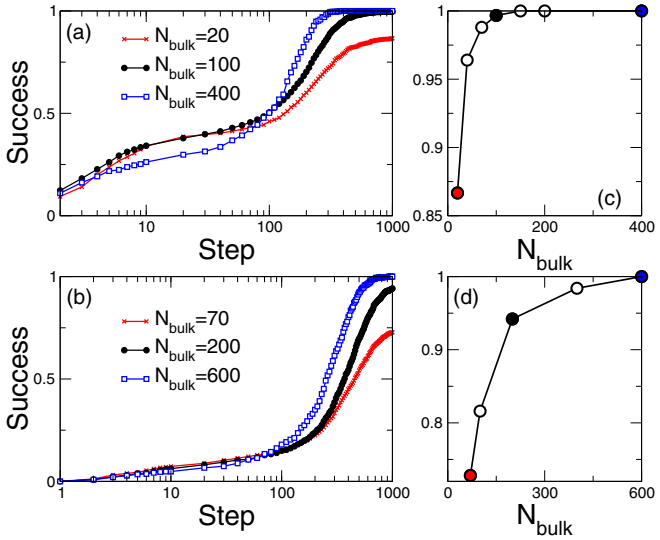

FIG. 5. Learning performance as a function of the middle layer size. Results show the success as a fraction of networks that learn a random input-output association map after a given number of correction steps for (a) $N_{\text{in}} = N_{\text{out}} = 3$ and $N_{\text{bulk}} = 20$, 100, and 400 and (b) $N_{\text{in}} = N_{\text{out}} = 4$ and $N_{\text{bulk}} = 70$, 200, and 600. Also shown is the success at 1000 steps as a function of N_{bulk} for (c) $N_{\text{in}} = N_{\text{out}} = 3$ and (d) $N_{\text{in}} = N_{\text{out}} = 4$. In (c) and (d) the colored symbols correspond to the results in (a) and (b) obtained with the respective N_{bulk} values. All results are averages over at least 500 network realizations, yielding values of standard errors smaller than the symbols size (standard error of the mean approximately equal to 0.025).

memristor model [12,13], with parameters fixed to represent the behavior of the first reported memristor [10,17], are shown in the Appendix. This similarity suggests that the performance of the learning algorithm presented here does not depend strongly on the details of the memristors used.

The implementation of the training algorithm shows also that the memristor network learns a series of maps in a way similar to that exhibited by the earlier neuronal model [3,4]. This can be seen in the example of Fig. 6, which shows the evolution of the network with $N_{\text{in}} = 4$, $N_{\text{bulk}} = 200$, and $N_{\text{out}} = 4$. The network is trained in one of the labeled maps until eventually the error is zero, at which point it starts being trained on a different map and so on. Notice the resemblance to the results in Fig. 1, which suggests that the training strategy proposed here is capturing the essence of the learning algorithm of Refs. [3,4].

Another distinctive property of the proposed training strategy is the fact that the memristive networks are robust to perturbations of the device properties. This alterations can be seen, for instance, as changes in resistance, which in real networks can be due to volatility, defects, etc. As an example, we plot in Fig. 7 the evolution of a network ($N_{\text{in}} = N_{\text{out}} = 4$ and $N_{\text{bulk}} = 200$) which, after learning the identity map, is periodically perturbed. It can be seen that after each perturbation, the network recovers to null error learning the map in a few additional steps. This ability is not surprising given the fact that the network proceeds with the perturbation in the same way as during the usual learning process.

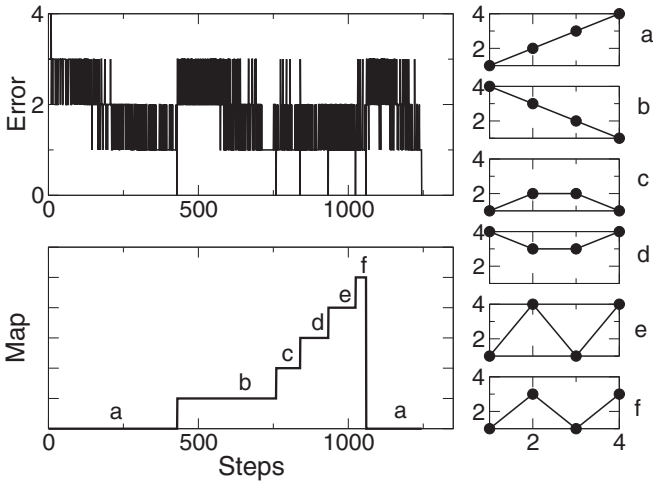

FIG. 6. Typical evolution of the training of a network learning seven successive maps for $N_{\text{in}} = N_{\text{out}} = 4$, $N_{\text{bulk}} = 200$, and the other parameters the same as in Fig. 5. The upper main panel shows the error (Hamming distance from the desired to the current output) as a function of training steps. (a)–(f) The input-output maps are depicted and presented sequentially as indicated in the lower main panel.

To gain insight into the dynamics of the memristor network resistances during learning of successive maps, a network with $N_{\text{in}} = N_{\text{out}} = 3$ and $N_{\text{bulk}} = 400$ is trained to learn all the possible $N_{\text{out}}^{N_{\text{in}}} = 27$ maps. After all maps are learned, the network location of each memristor is randomly shuffled (preserving their resistance values). After that, the network is retrained to learn the same $N_{\text{out}}^{N_{\text{in}}}$ maps.

In Fig. 8(a) we show the histogram of resistances, at the beginning of the simulation, after learning all possible maps once (labeled 27), after shuffling and relearning all maps four

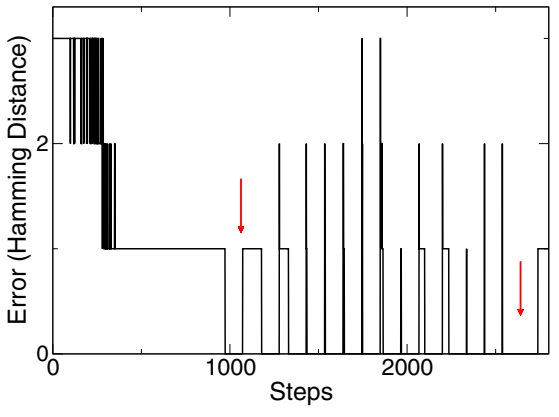

FIG. 7. Example of the network recovery after a single perturbation for $N_{\text{in}} = N_{\text{out}} = 4$, $N_{\text{bulk}} = 200$, and the other parameters the same as in Fig. 5. We plot the error (i.e., the Hamming distance from the desired to the current output) as a function of the training steps for a network which is learning the identity map. One hundred steps after the map is learned, 10% randomly chosen memristors are perturbed by increasing their resistances by 5%. The first downward arrow indicates the first perturbation, while the second downward arrow denotes the 14th perturbation (which did not increase the network error).

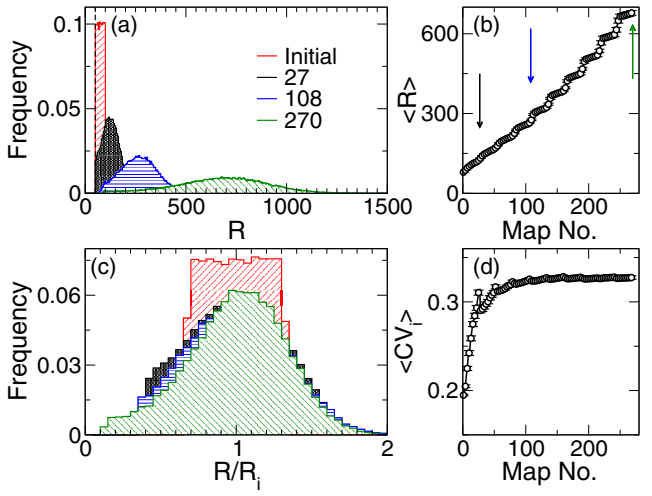

FIG. 8. Evolution of the distribution of resistance values after repeated relearning of the same set of maps for a network with $N_{in} = N_{out} = 3$, $N_{bulk} = 400$, and other parameters values as in Fig. 5. (a) Resistance histogram at the beginning of the simulation, after learning all possible maps once (labeled 27), after shuffling and relearning all maps four times (labeled 108), and after shuffling and relearning all maps ten times (labeled 270). The dashed vertical line shows the minimum possible value $R_{min} = 50$. (b) Average resistance as a function of the number of learned maps. Arrows indicate the maps used for the histograms in (a). (c) Same data as in (a) after normalizing each memristor's resistance by the average value of the whole network. (d) Coefficient of variation of the memristor's resistance as a function of the number of learned maps. Results are calculated for a single network and then averaged over 25 network realizations. The bin size (a) $\Delta R = 5$ and (b) $\Delta R = 0.05 R_i$.

times (labeled 108), and after shuffling and relearning all maps ten times (labeled 270). In Fig. 8(b) we show the evolution of the average resistance as a function of the number of learned maps. As expected from the nature of the training algorithm, resistances can only grow when performing correction steps. Moreover, in Fig. 8(a) the range of resistance values increases with the number of steps. We remove the effect of growing resistance by normalizing with the average resistance of the ith network R_i. In Fig. 8(c) we plot a histogram of the normalized resistances, where each resistance value is divided by the average resistance of the network. Finally, in Fig. 8(d) we show the coefficient of variation $\langle CV \rangle$ of a single network, computed as the standard deviation divided by the mean value of all resistances in the network. After repeated learning, the distribution tends to a Gaussian distribution, approaching $\langle CV \rangle \sim \frac{1}{3}$.

IV. DISCUSSION

Several learning strategies for memristor networks have been proposed recently [18–21]. For problems with time-dependent inputs, one strategy consists of using reservoir computing [22], where the inputs are connected to a reservoir network composed of interconnected time-dependent elements (whose properties continue evolving after a stimulus is applied), such as volatile memristors. The response of the network is then classified through an output layer, which needs to be trained. Reservoir computing has also been applied to time-independent inputs that are reencoded as time-dependent ones [20]. While this kind of network differs from those studied here, an error penalization learning mechanism, such as the one studied here, may be useful for training the output layer of this scheme.

For time-independent inputs, several learning strategies considering nonvolatile memristors (as done here) have also been proposed. Most of them are inspired by machine learning algorithms, such as gradient-descent training (see, e.g., [18,21]), where the value of each resistance needs to be known and modified at each correction step. As an alternative, a random weight change algorithm, which does not require knowing the precise values of all elements, was recently proposed for training a network where each synapse is composed of four memristors and several transistors [19].

Similarly, here we have described a training procedure allowing a simple network of memristors to learn any arbitrary input-output association map. This is achieved without any detailed information about the inner structure of the layers. The method is inspired by a learning algorithm proposed about 20 years ago [3,4]; however, there are some differences among them.

First, in the original model, the active neurons transmitted their activity to just one neuron on the following layer, while here the current passing through each node is determined by Kirchoff's laws. Probably, the inclusion of diodes or transistors (such as in Refs. [19,20]) may generate dynamics which are closer to the original proposal. The study of this possibility is an interesting avenue for future research.

Second, in the correction step for the toy model, only the (two) intervening connections are punished. Here, instead, we apply V_{write} over the input and output nodes in such way that the correction voltage difference (and consequently the correction) over each memristor in the network is proportional to the voltage involved in the wrong answer. In this way, the correction is performed without the need to measure or to have control over the middle layer: The learning method does not need to control individual memristors and requires only access to read and/or perturb two nodes from the (arbitrarily large) networks at any given time (consider, as an example, the network of Fig. 8, which has 2400 memristors and only three input nodes plus three output nodes need to be considered in the present learning method).

Despite the extreme simplicity of the approach, it is demonstrated that an iterative reading of the current flowing between two points of the network, and its eventual perturbation, can modify the overall network connectivity until arriving at one of the possible solutions. By design, the learning is robust against different types of perturbations, including differences and fluctuations in the memristor parameters, noise, and defects, as well as distribution of polarities (see the Appendix). Although the present results are limited to numerical simulations, in case of being implemented in hardware, the method is easily scalable to arbitrarily large network sizes.

Note that the memristor network is able to learn despite lacking an important feature of its biological neural counterpart: the spikes. That is not a limitation, because as noted in the Introduction, in this kind of network, the only role for the

spikes would be to propagate the information from the input node(s) through the network to some output node(s). Instead, in the present implementation, this is also achieved, but by the current flow from an external battery. Thus, knowing which input node is connected, the current flowing through the network will be reflected on the value of the output current at a given node. In this context, our approach is a simple solution which does not require any implementation of any spiking mechanism. Obviously, the absence of additional electronics to implement neurons becomes very relevant when considering a hardware implementation of this concept. Concerning hardware, the proposed learning method benefits from variability in the network properties, therefore not requiring precise control of the memristor parameter during manufacturing.

Several analytical results have been provided for the original model [5,6]. For instance, it has been shown that a geometric transition from no learning to learning takes place at $N_{\text{bulk}} = N_{\text{in}} \times N_{\text{out}}$. While our numerical simulation results show similarities to the original learning algorithm, it would be useful to verify whether and how these results are valid here. Also, extending other analytical approaches, such as the work by Caravelli *et al.* [23,24], to the system studied here, should give a deeper understanding of how learning in this memristor network takes place.

In this paper we limited ourselves to the presentation of the most fundamental aspects of the results. As a salient feature, we reported how a single network can learn many maps, and this causes an expansion of the distribution of their resistances, possibly due to the multiplicity of solutions to learn a single map. Nonetheless, a few caveats must be mentioned. First, we consider adversarial situations for the algorithm, concerning some simple variations on the approach, where changes in the initial condition of memristor conductances and polarity as well as minor changes in the type of correction step are described. These studies are presented in the Appendix. Second, we did not expand the discussion of the types of problems that the present approach can solve. This issue would require extensive numerical simulations and it seems to deserve being explored on a hardware implementation, since it will work tens of orders of magnitude faster than any of our current numerical simulations. Finally, we expect the approach to be useful on a variety of network topologies including less ordered systems such as a random network of nanowires [25] whose conductivity can be varied by applying a voltage difference among pairs of points in the network. The algorithm may also be useful on other structures [26] as long as a correction mechanism increasing resistance over undesired paths can be generated.

V. CONCLUSION

In summary, we have introduced an algorithm able to train a memristor network to learn arbitrary associations. Robust results for its performance were demonstrated using numerical simulations of a network of voltage-controlled memristive devices. Given the design principles, the results suggest that its implementation in hardware would be straightforward, being scalable and requiring very little peripheral computation overhead.

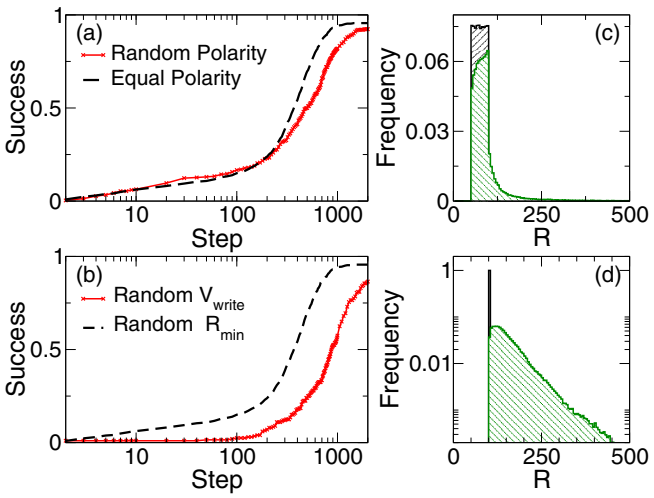

FIG. 9. Performance of the approach using a random distribution of memristor polarities or random V_{write} correction values. (a) Success as a function of step number for networks where memristors polarities are chosen at random . (b) Success as a function of step number for equal initial conditions $R_{\text{min}} = 100$ and correction voltages V_{write} uniformly drawn from 0.15 to 0.3. Also shown is the initial and final resistance density distribution for the case of using (c) random polarity and (d) random correction V_{write}. For comparison, in (a) and (b) the dashed line reproduces the results presented in Fig. 4(b) ($N_B = 200$, black circles). In (c) and (d), the bin width $\Delta R = 5$. Results are averaged over at least 100 network realizations using, in all cases, $N_{\text{in}} = N_{\text{out}} = 4$ and $N_{\text{bulk}} = 200$.

ACKNOWLEDGMENTS

This work was partially supported by Grant No. 1U19NS107464-01 from NIH BRAIN Initiative (USA) and CONICET (Argentina).

APPENDIX

1. Miscellaneous observations

Some special cases are described here, including different initial conditions of the memristor parameters and variations on the implementation of the correction step, noting that all the results remain valid despite these changes. Figure 9(a) shows the results of simulations where the polarity of the memristors is distributed randomly. As a comparison, we plotted (with dashed line) the data already in Fig. 5(b) corresponding to equal memristor polarity. Figure 9(c) shows the changes in the R distribution before and after the learning process. These results suggest that the distribution of polarities produces only minor changes in the overall performance of the approach.

Then we explored how the lack of variability of the memristor's properties may affect the performance, by starting the simulation with identical resistances ($R = 100$) for all memristors and using random V_{write} values in the correction steps (uniformly distributed from 0.15 to 0.3) while keeping the other parameters β and $V_{\updownarrow}$ randomly distributed. The results are shown in Fig. 9(d). It is apparent that the network learns approximately in the same manner as when starting with random initial conditions for R, except that it takes additional

steps to reach comparable success rates. Probably these additional correction steps are trivially related to the time needed to generate a minimal dispersion of the R values, needed for the approach to work. Thus, the manufacturing variability of the memristor properties expected in an experimental setup will not be disadvantageous. In Fig. 9(d) the initial (i.e., $R = 100$) and final distributions of resistances for this case are shown, showing the resulting broad R distribution, after the map is learned.

2. Alternative memristor model

We have reproduced some results in the main text using a different memristor model, known as the boundary condition memristor (BCM). This model was proposed in Ref. [12]. In Ref. [13] it was shown that the BCM model reproduces the current-voltage characteristics of Pickett's model [17] (which explains the dynamics of the first reported memristor, based on TiO_2 nanofilms [10]) closer than other alternative descriptions, when the parameters are chosen adequately.

The equations for a single BCM can be written as a function of a state parameter ω as

$$I = \frac{V}{R}, \tag{A1}$$

$$R = R_{\max} - \frac{\omega}{D}(R_{\max} - R_{\min}), \tag{A2}$$

$$\frac{\partial \omega}{\partial t} = \frac{\mu R_{\min}}{D} I f_B(\omega, V), \tag{A3}$$

where $D = 10$ nm is the assumed width of the memristor ($0 \leqslant \frac{\omega}{D} \leqslant 1$) and f_B may have three different values ($a > 0, b > a, 0$) depending on four non-negative constants v_{t0}, v_{t1}, v_{th0}, and v_{th1} and the values of ω and V: For $0 < \frac{\omega}{D} < 1$,

$$f_B(\omega, V) = \begin{cases} a & \text{if } -v_{t1} \leqslant V \leqslant v_{t0} \\ b & \text{otherwise.} \end{cases} \tag{A4}$$

For $\frac{\omega}{D} = 1$ (minimum resistance),

$$f_B(\omega, V) = \begin{cases} b & \text{if } V < -v_{th1} \\ 0 & \text{otherwise.} \end{cases} \tag{A5}$$

For $\frac{\omega}{D} = 0$ (maximum resistance),

$$f_B(\omega, V) = \begin{cases} b & \text{if } V > v_{th0} \\ 0 & \text{otherwise.} \end{cases} \tag{A6}$$

The parameters which more closely reproduced Pickett's result, having $D = 10$ nm, $\mu = 10^{-16}$ m^2 V^{-1} s^{-1}, $R_{\min} = 10^3$ Ω, and $R_{\max} = 10^4$ Ω fixed, were $a = 0.1494$, $b = 1.6182$, $v_{t0} = 0.915$ V, $v_{t1} = 1.3048$ V, $v_{th0} = 4.7404$ V, and $v_{th1} = 2.4629$ V (a and b are unitless) (see Ref. [12]).

Equations (A2) and (A3) can be rewritten in terms of Eq. (2) by setting $F(R, V) = \frac{\mu(R_{\max} - R_{\min})R_{\min}}{D^2} \frac{V}{R} f_B(\omega(R), V)$. The function $F(R, V)$ for the BCM model is shown in Fig. 10. Note that the BCM model shows some differences from the model in the main text (the BMS). The most important one is that now the correction function F depends on the current $I = V/R$. This means that, intuitively, in the first model, highest resistance values would tend to show larger voltage differences and thus to have stronger corrections (for instance, when connected in series with another resistance), while here the lowest resistance values will tend to have higher corrections. Also, the model presents four different voltage thresholds and resistance evolution even with small voltages applied (for $a > 0$).

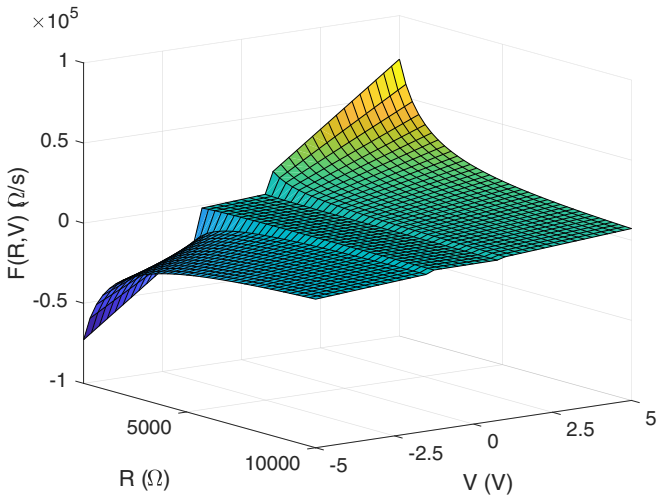

FIG. 10. The BCM behavior as a function of resistance and applied voltage, using the parameters proposed in Ref. [12]: $D = 10$ nm, $\mu = 10^{-16}$ m^2 V^{-1} s^{-1}, $R_{\min} = 10^3$ Ω, $R_{\max} = 10^4$ Ω, $a = 0.1494$, $b = 1.6182$, $v_{t0} = 0.915$ V, $v_{t1} = 1.3048$ V, $v_{th0} = 4.7404$ V, and $v_{th1} = 2.4629$ V.

We have reproduced the results shown in the main text using BCMs whose evolution is given by Eqs. (A3)–(A6) and the parameters listed above, except $R_{\min}$, which is randomly chosen between 500 and 1000 Ω for each memristor. The algorithm parameters are now $V_{\text{read}} = 0.0001$ V and $V_{\text{write}} = -5$ V and the time over which the correction voltage is applied is set to $\Delta t = 1$ ms. Results similar to those presented in main text are shown in Fig. 11. A detailed analysis of the performance of this method as a function of the values of memristor parameters (v_{t0}, v_{t1}, v_{th0}, v_{th1}, $R_{\min}$, $R_{\max}$, a, and b), including noisy parameter distribution or the parameters of the learning algorithm (V_{read}, V_{write}, and Δt), exceeds the scope of this Appendix.

3. Pseudocode

The computer codes used for generating numerical simulation results have been uploaded to online repositories [14], as stated on the main text. Here we summarize the code structure.

Before network simulation starts, variables are defined.

```
** Define variables **
* Number of input, bulk and output nodes: *
N_in, N_bulk, N_out
* Algorithm Voltages: *
V_read=0.00001 V_write=-0.2
* Node voltage vector variables: *
V_in(N_in), V_bulk(N_bulk), V_out(N_out)
* Resistance values and currents: *
R_In-Bulk(N_in,N_out),
R_Bulk-Out(N_bulk,N_out),
I_In-Bulk(N_in,N_out),
I_Bulk-Out(N_bulk,N_out)
```

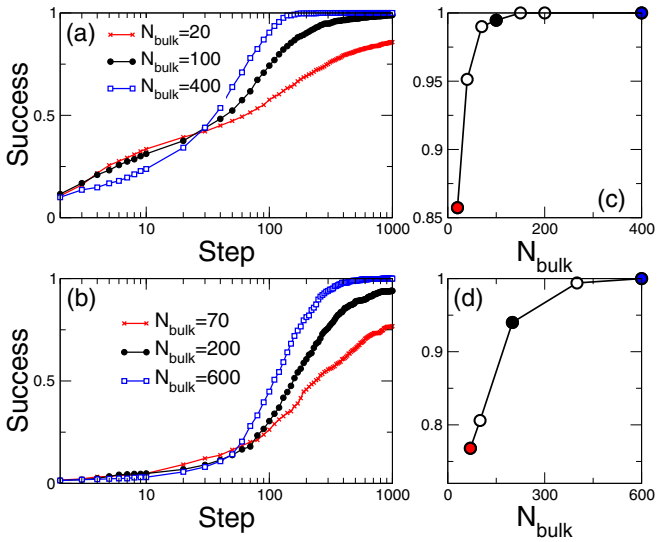

FIG. 11. Learning performance as a function of the middle layer size for BCMs. Results show the success as the fraction of networks that learn a random input-output association map after a given number of correction steps for (a) $N_{\text{in}} = N_{\text{out}} = 3$ and $N_{\text{bulk}} = 20$, 100, and 400 and (b) $N_{\text{in}} = N_{\text{out}} = 4$ and $N_{\text{bulk}} = 70$, 200, and 600. Also shown is the success at 1000 steps as a function of N_{bulk} for (c) $N_{\text{in}} = N_{\text{out}} = 3$ and (d) $N_{\text{in}} = N_{\text{out}} = 4$. In (c) and (d) the colored symbols correspond to the results in (a) and (b) obtained with the respective N_{bulk} values.

```
* Define other auxiliary variables *
```
Each memristor parameter is chosen.
```
** Generate memristor parameters **
* beta, V_threshold, R_min-plx-sol-plxmax *
for i=1,N_in; for j=1,N_bulk
beta_In-Bulk(i,j)=random(0.8-1)
VT_In-Bulk(i,j)= random(0.05-0.1)
R_min_In-Bulk(i,j)=random(50-100)
R_max_In-Bulk(i,j)=5000
end for (j); end for (i)
* Do the same for beta_Bulk-Out, *
*VT_Bulk-Out, and Rmin-plx-sol-plxmax_Bulk-Out*
```

A learning task is selected. In this case, a random input-output map is selected.

```
** Generate random Map **
for i=1,N_in
input-output-MAP(i)=random_integer(1,N_out)
end for(i)
```

In the main routine, up to 1000 learning steps are performed. Within each learning step, up to $n_s = 80$ correction steps are applied. The main routine uses three subroutines: READ, CORRECT, and COMPUTE ERROR.

```
*** Main Routine ***
for learning_step=1,1000
input_node=random_integer(1,N_in)
for correct<n_s
call READ
if (output_node =
=input-output-MAP(input_node))
FINISH current learning_step
else
call WRITE(input_node,output_node)
end for (correct)
COMPUTE ERROR
if (error==0) SUCCESS, EXIT.
end for (learning_step)
END
```

The following is the pseudocode for READ, which requires the calculation of currents and voltages following Kirchoff's equations.

```
READ routine(input_node)
for k=1,N_out
Solve the circuit equations˜when
V_write is applied among input_node
and the k-th output node.
end for (k)
Report output_node as the one
with maximum current.
```

The following is the pseudocode for WRITE.

```
WRITE routine(input_node,output_node)
for time=1,5
Solve the circuit equations˜when
V_read is applied among input_node
and output.
Calculate voltage difference on
each memristor.
Update resistances (using
memrisor equations).
end for(time)
```

The following is the pseudocode for COMPUTE ERROR.

```
COMPUTE ERROR routine
error=0
for i=1,N_in
call READ(i)
if (output_node -plx-sol-plx= input-output
  -MAP(i))
error\ensuremath{+}\ensuremath{+}
end for(i)
```

[1] D. Marković, A. Mizrahi, A. Querlioz, and J. Grollier, Nat. Rev. Phys. **2**, 499 (2020).

[2] P. Bak, *How Nature Works: The Science of Self-Organized Criticality* (Springer Science, New York, 1996).

[3] D. R. Chialvo and P. Bak, Neuroscience **90**, 1137 (1999).
[4] P. Bak and D. R. Chialvo, Phys. Rev. E **63**, 031912 (2001).
[5] J. Wakeling, Physica A **325**, 561 (2003).
[6] J. Wakeling, Physica A **340**, 766 (2004).
[7] M. Brigham, Self-organised learning in the Chialvo-Bak model, M.Sc. thesis, University of Edinburgh, 2009.
[8] L. O. Chua, IEEE Trans. Circuit Theory **18**, 507 (1971).
[9] F. Caravelli and J. P. Carbajal, Technologies **6**, 118 (2018).
[10] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, Nature (London) **453**, 80 (2008).
[11] D. Biolek, M. Di Ventra, and Y. V. Pershin, Radioengineering **22**, 945 (2013).
[12] F. Corinto and A. Ascoli, IEEE Trans. Circ. Syst. I **59**, 2713 (2012).
[13] A. Ascoli, F. Corinto, V. Senger, and R. Tetzlaff, IEEE Circ. Syst. Mag. **13**, 89 (2013).
[14] The computer code to reproduce the following results can be downloaded from https://github.com/DanielAlejandroMartin/Memristor121 (version 1.0). A complementary library written in PYTHON is available at https://gitlab.com/kakila/memnet and further considerations regarding potential memristive implementation are available at https://kakila.gitlab.io/mistake-learning/.
[15] D. Querlioz, P. Dollfus, O. Bichler, and C. Gamrat, *Proceedings of the IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), San Diego, 2011* (IEEE, Piscataway, 2011), pp. 150–156.
[16] Q. Xia and J. J. Yang, Nat. Mater. **18**, 309 (2019).
[17] M. D. Pickett, D. B. Strukov, J. L. Borghetti, J. J. Yang, G. S. Snider, D. R. Stewart, and R. S. Williams, J. Appl. Phys. **106**, 074508 (2009).
[18] R. Hasan, T. M. Tarek, and C. Yakopcic, Microelectron. J. **66**, 31 (2017).
[19] C. Yang, S. P. Adhikari, and H. Kim, IEEE Trans. Circ. Syst. I **66**, 3906 (2019).
[20] R. Midya, Z. Wang, S. Asapu, X. Zhang, M. Rao, W. Song, Y. Zhuo, N. Upadhyay, Q. Xia, and J. J. Yang, Adv. Intell. Syst. **1**, 1900084 (2019).
[21] D. Soudry, D. Di Castro, A. Gal, A. Kolodny, and S. Kvatinsky, IEEE Trans. Neural Netw. Learn. Syst. **26**, 2408 (2015).
[22] J. P. Carbajal, J. Dambre, M. Hermans, and B. Schrauwen, Neural Comput. **27**, 725 (2015).
[23] F. Caravelli, Phys. Rev. E **96**, 052206 (2017).
[24] F. Caravelli, F. C. Sheldon, and F. L. Traversa, Sci. Adv. **7**, eabh1542 (2021).
[25] H. G. Manning, F. Niosi, C. G. da Rocha, A. T. Bellew, C. O'Callaghan, S. Biswas, P. F. Flowers, B. J. Wiley, J. D. Holmes, M. S. Ferreira, and J. J. Boland, Nat. Commun. **9**, 3219 (2018).
[26] S. Fostner, R. Brown, J. Carr, and S. A. Brown, Phys. Rev. B **89**, 075402 (2014).