An approximate backpropagation learning rule for memristor based neural networks using synaptic plasticity

D. Negrov[a,*], I. Karandashev[a,b], V. Shakirov[a,b], Yu. Matveyev[a,c], W. Dunin-Barkowski[a,b], A. Zenkevich[a,c]

[a] *Laboratory of Functional Materials and Devices for Nanoelectronics, Moscow Institute of Physics and Technology, Dolgoprudny, Russian Federation*
[b] *Scientific Research Institute for System Analysis, Russian Academy of Sciences, Moscow, Russian Federation*
[c] *National Research Nuclear University MEPhI, Moscow, Russian Federation*

ARTICLE INFO

ABSTRACT

We describe an approximation to backpropagation algorithm for training deep neural networks, which is designed to work with synapses implemented with memristors. The key idea is to represent the values of both the input signal and the backpropagated delta value with a series of pulses that trigger multiple positive or negative updates of the synaptic weight, and to use the min operation instead of the product of the two signals. In computational simulations, we show that the proposed approximation to backpropagation is well converged and may be suitable for memristor implementations of multilayer neural networks.

1. Introduction

Recent advances in machine learning have provided the solutions to many problems, which seemed insurmountable in the past. New approaches based on deep and recurrent artificial neural networks (ANN) are very efficient in dealing with pattern recognition, visual objects detection, speech recognition, signal restoration and prediction, etc. [1]. Deep feed forward networks excel at these tasks because their design can be general enough to solve many natural classification tasks, but their main feature is a funnel-like error landscape, which opens a possibility to apply computationally efficient learning methods, such as stochastic gradient descent.

Nowadays, the computations with ANNs are performed mostly using conventional computation architectures by simulation on general purpose processors or graphics cards which are quite efficient at matrix algebra being the core of feed forward deep networks. However, due to the limited level of parallelism this approach is quite inefficient in terms of the ratio between the computational speed and the power consumption. Consequently, it hinders the application of these methods in areas, where the available power is limited, such as mobile devices or autonomous robotic platforms. A further step in applying deep learning neural networks to real-life problems would depend on their implementation in hardware which would use accelerators specifically built to perform neural computations and employ the most possible parallelism. The optional solution here is to create a system which mimics the structure of the network under consideration by implementing neurons and their interconnections directly in hardware. In this case, the computational fabric of the device can be made of blocks representing neurons, which are interconnected with synapses with weights implemented as distributed memory, similar to the biological neural tissue.

One of the major problems hindering the design of artificial neural network hardware is the storage of synaptic weights. The distributed approach to the storage makes use of dynamic RAM (DRAM) very inconvenient due to the requirement of constant refresh operations arising from the leakage of charge [2].

Among the approaches existing today, the one coming close to take full advantage of huge parallelism of ANNs is an FPGA-based implementation, such as described in [3]. Such implementations rely on the array of specialized processing units, tailored to compute outputs of neural network layer. Being quite efficient, this approach, however, suffers from two drawbacks. Firstly, the distributed RAM on modern FPGA is implemented using static RAM arrays, which have low density, making the available memory limited. Hence, in this case, the implementation of large-scale deep network might require constant exchange with the external DRAM, which is inconvenient. Secondly, despite being quite universal, FPGAs suffer from low connectivity and limited availability of multiply-and-accumulate modules, and therefore they must be shared among different neurons. Moreover, the incorporation of on-line learning in such devices generally requires supervision from the external CPU.

---

* Corresponding author.
  E-mail address: dmitriynegrov@gmail.com (D. Negrov).

Over the last several years, a variety of memristive devices has been discovered. These devices are the resistors with the conductance controlled by the current or voltage previously applied to them. In this way, they represent the class of non-volatile memory devices with effective continuum of memory states, which can be scaled down to a ten nanometers. These properties of memristors have opened an opportunity to implement a hardware neural network as an array of cores, which consist of a bank of presynaptic and postsynaptic neurons, made with the use of conventional CMOS technology, and interconnected with an array of memristive devices. Such arrays can be arranged as cross-bar arrays, thus providing an efficient storage medium [4–8], useful for the effective modeling of physiological processes [9] and the implementation of wide-spread types of neural networks, such as cellular neural networks (CNN) [10]. The comprehensive survey of memristors and memristor-based techniques with the necessary references to previous works are given in the seminal paper of the field's pioneer Chua [11].

For the hardware implementation of hybrid CMOS/memristor neural networks, one of the biggest challenges is training [12]. Even simplest training algorithms, such as error backpropagation [13], are difficult to implement at the circuit level. It is much easier to implement these algorithms off the chip and then transfer the training results [14]. Another approach is to train only the output layer of the network, which can be accomplished using a simple least-mean-squares learning algorithm [15,12,16,17].

In [2], an alternative Random Weight Change learning algorithm is used. Although its convergence is theoretically proven, it appears to be very slow. For each synaptic weight it uses four memristors in a bridge configuration. Besides, there is an independent random number generator in each model neuron. The scheme in [18] uses standard backpropagation scheme and two memristors for each synapse. Unfortunately, the learning is implemented there using an external training unit, and this part of the network learning essentially uses MATLAB, because it apparently does not have a simple implementation in circuitry, while the other part of the modeling is performed via SPICE simulator. A significant advantage of the work [19] is the clearly formulated problem of *non-local calculation of the signal and error product at the opposite contacts of a memristor*. However, the particular solution proposed by the authors critically depends on the linearity of the memristor resistance as a function of both the amplitude and the duration of the passed current. This condition does not yet hold for the majority of the current memristor implementations.

Meanwhile, there has been a large number of works emulating a so-called synaptic plasticity in memristor devices (e.g., [8,20]). They are aimed at demonstrating effects similar to the plasticity of biological synapses, such as short and long term potentiation/depression and spike-timing-dependent synaptic plasticity. Despite the fact that these models are biologicaly inspired, there is no clear idea of how to use them for the implementation of a learning procedure which would be adequate for solving machine learning problems.

In this paper, we aim to bridge this gap and propose a modification of backpropagation algorithm which allows to circumvent the described difficulties using the learning rule similar to spike-timing-dependent plasticity in synapses. Strictly speaking, we do not give a hardware neural network implementation scheme with a memristor crossbar, but we thoroughly describe the method of pulse representation of signals and the absmin operation used instead of standard product, and also perform the modeling experiments in MATLAB environment.

The remainder of the paper is organized as follows: in Section 2.1, we give the problem statement. In Section 2.2 the backpropagation algorithm is briefly described. In Section 2.3 we propose our method of signal representation and the operation used instead of standard product. Section 3 describes the test problem of handwritten digits recognition and parameters of modeling experiments. In Section 4 the results are given. Finally, the discussion of the proposed methods and the obtained results is given in Section 5.

2. Material and methods

2.1. Problem statement

In this work, we do not consider any specific scheme of multilayer neural network implementation on a memristor crossbar. However, we assume that a neuron is a device which sums the currents incoming from other neurons. These currents are defined by the conductance of memristors connecting the neurons, i.e. the synapse weight between the two neurons is the conductance of the interconnecting memristor. Thus, in order to set one weight, one memristor is enough, and the memristor crossbar forms the matrix of connections among all neurons.

The problem of neural networks implementation, irrespective of whether it is implemented in software or on the chip, can be divided into two parts: the inference problem and the learning of the network. In this paper, we do not consider the problem of inference, because the forward signal propagation through the multilayer network is reduced to currents passing through memristors crossbar, which is the direct consequence of the Ohm's law.

The issue considered in this work concerns the learning of the network, where the weight (connection) of synapses between neurons is represented by the memristor conductance. According to the back-propagation algorithm (see Section 2.3), the weight update should be proportional to the product of two variables: the forward signal x_i and backward error signal δ_j. The values x_i and δ_j are calculated in different neurons (Fig. 1), located in different layers of the neural network. It is desirable for them to interact only via the synaptic connection, and this interaction constitutes the main difficulty for hardware implementation of backpropagation learning rule in multilayer neural networks. In Section 2.3, we will show how in this case the signals x_i and δ_j should be represented in neurons in order to learn the network.

2.2. Backpropagation

In this section, we give a short sketch of the backpropagation technique (the readers familiar with the backpropagation learning rule can skip it and immediately move to the next section).

The task of a multilayer perceptron is to get the desired output to the inputs of certain types. With this goal in mind, the learning of perceptron is performed. The pairs of inputs and the desired outputs are loaded to the scheme and the error of the response is determined. The parameters of the scheme (the weights of inter-neuronal connections) are changed with each load, so as to diminish the difference between the desired and the real output (Fig. 1).

Let us denote the desired output of the neurons as t_j, and its actual output as y_j. We will further use two measures of network error — the mean−square−error ϵ_{MSE} and the cross−entropy ϵ_{CE}:

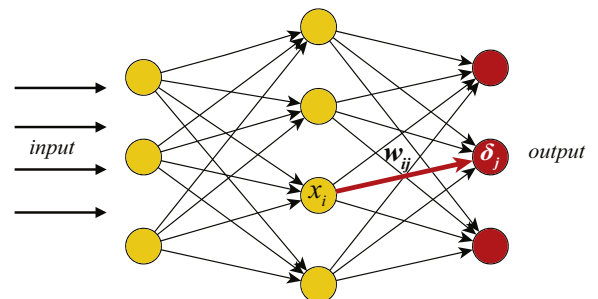$$\epsilon_{MSE} = \sum_j (y_j - t_j)^2 \tag{1}$$

Fig. 1. The scheme of a multi-layer perceptron.

$$\epsilon_{CE} = - \sum_j t_j \ln y_j + (1 - t_j)\ln(1 - y_j) \tag{2}$$

Let w_{ij} be the weight of the connection between j-th neuron of the last layer and i-th neuron of the previous layer (Fig. 1). The derivative of error with respect to this variable can be written as:

$$\frac{\partial \epsilon}{\partial w_{ij}} = \frac{\partial \epsilon}{\partial y_j}\frac{\partial y_j}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}} = \dot{\epsilon}(y_j)\dot{f}(z_j)x_i = \delta_j x_i \tag{3}$$

where

$$z_j = \sum_i w_{ij}x_i, \; y_j = f(z_j), \; \delta_j = \dot{E}(y_j)\dot{f}(z_j)$$

Let the output neurons have a sigmoidal activation function $f(z) = \frac{1}{1 + \exp(-z)}$, and we use Eq. (1) for the error calculation. Then, for the Eq. (3) we get:

$$\frac{\partial \epsilon}{\partial w_{ij}} = (y_j - t_j)x_i. \tag{4}$$

Let us assume that the layers of neurons are numbered from 0 to L, and the weights of connections from the layer $i - 1$ to the layer i have the upper index i. Then, for all intermediate layers we can obtain expressions similar to (4).

$$\frac{\partial \epsilon}{\partial w_{ij}^{(k)}} = x_i^{(k-1)}\delta_i^{(k)}, \tag{5}$$

where $\delta_j^{(L)} \equiv \delta_j = (y_j - t_j)$, x_i^0 are input signals and

$$\delta_j^{(k-1)} = \sum_i w_{ji}^{(k)}\delta_i^{(k)}\dot{f}(z_j^{(k-1)}), \tag{6}$$

so that error terms $\delta_j^{(k)}$ are propagated backwards using transposes of weight matrices and the derivative of the activation function. Thus, the back-propagation algorithm is the pair of Eqs. (5) and (6) with an additional rule for the weight update. Fig. 2 shows how signal propagates forward and backward in the network.

Alternatively, the error can backpropagate using another weight matrix $B^{(k)}$, which is different from $W^{(k)}$ and is kept constant during the learning process. Such method was devised in the work [21], where elements of matrix B were chosen randomly. Formula (6) in this case can be rewritten as:

$$\delta_j^{(k-1)} = \sum_i b_{ji}^{(k)}\delta_i^{(k)}\dot{f}(z_j^{(k-1)}), \tag{7}$$

2.3. Signal representation and multiplication using memristors

Despite the fact that the following procedure is quite general, we are aiming to use it for the implementation of neuromorphic systems employing metal oxide based memristors. This class of memristors has a very non-linear behavior, and their characteristic feature is the presence of a voltage "dead zone", i.e. up to the certain voltage levels, the memristor resistance state is kept unchanged. Moreover, the gradual change of the state in such device can be achieved by applying a voltage pulse train — the property which has been demonstrated in the works reporting on synaptic plasticity effects [8,20]. This property paves the way for the implementation of a learning procedure in memristor matrices using only local operations as follows.

In order to use the formula (5) in optimization algorithms using gradient descent, the weight update should be proportional to the product of two variables:

$$\Delta w_{ij} \propto x_i \delta_j. \tag{8}$$

The values x_i and δ_j are obtained in different neurons (Fig. 1), located in different layers of the neural network. It is desirable for them to interact only via the synaptic memristor. To deal with this problem, we use the following approach: taking high non-linearity of metal oxide memristors as an advantage, we choose such a voltage $u_+ > V_{on}$, which results in the change of the memristor state (resistance) (Fig. 3), while half of that value does not lead to any changes ($u_+/2 < V_{on}$). By applying voltage pulses with amplitudes u_+, one can change the synapse resistivity in small steps. Similarly, we choose voltage pulses of amplitude $u_- < V_{off}$ to change conductivity in the opposite direction.

Let us assume firstly that both of x_i and δ_j are positive. Their values can be represented by two series of pulses, both with the amplitude $u_+/2$, so that pulses for x_i an δ_j have opposite polarities and their number is proportional to the absolute values of the signals (Fig. 4). With such representation, the voltage drop across the memristor exceeds the threshold V_{on} only for those pulses, which simultaneously arrive to the opposite electrodes of the device.

According to this procedure, the weight change Δw_{ij} is proportional to the minimum between x_i and δ_j

$$\Delta w_{ij} \propto \min(x_i, \delta_j). \tag{9}$$

In case x_i and δ_j have opposite signs, Δw_{ij} should decrease. This can be achieved by changing the pulse polarity. In the general case, when x_i and δ_j can be either positive or negative, (9) should be modified to:

$$\Delta w_{ij} \propto sign(x_i\delta_j)\cdot\min(|x_i|, |\delta_j|). \tag{10}$$

In fact, the expression (10) approximates (8) quite satisfactory for the network learning purposes (see Section 3 below), as it yields correct direction for a gradient descent process. The reason for this can be seen in Fig. 5. Therefore, this update rule will be used below in computational experiments.

In principle, the learning cycle should be divided into four subcycles for each combination of signs of x_i and δ_j. However, in this work we consider only non-negatively valued activation functions, and thus $x_i \geq 0$. In this case, $sign(x_i\cdot\delta_j)$ depends only on δ_j and only two subcycles are required for the system to function.

The first subcycle for the positive value of $sign(x_i\cdot\delta_j)$ has been already described above. We further consider u_- to be the voltage below
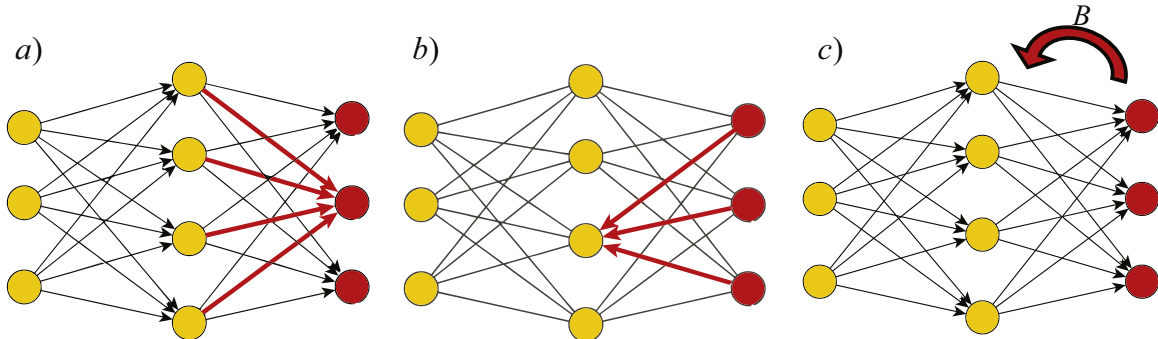

Fig. 2. Error back-propagation: (a) forward propagation of signals; (b) error propagation in the backward direction using the same bonds as in a, but in the opposite the direction; (c) error propagation using another fixed weight matrix, which is constant during learning process.
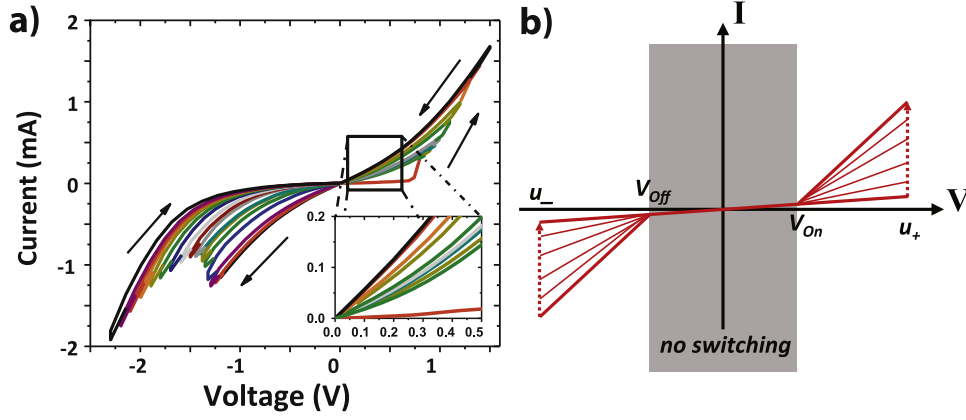
Fig. 3. (a) The experimental I-V curves corresponding to the resistive switching cycles of a Pt/Hf$_x$Al$_{1-x}$O$_{1-y}$/TiN memristor [22]; (b) the schematic drawing of the memristor switching cycles.

the lower writing threshold V_{off}, i.e. by applying it on the memristor one decreases its conductivity, and $V_{off} < u_-/2$. In order to implement the second phase of the learning cycle (when $\Delta w_{ij} < 0$), let us express the signals x_i and δ_j in the form of pulses with an amplitude $|u_-|/2$, as shown in Fig. 4(b). As a result, the overall bias on the memristor is negative when pulses coincide, and the weight of memristor will decrease.

In summary, the learning procedure consists of two phases, which go successively one after another and are illustrated in Fig. 4 (a) and (b). During the first phase, the i-th neuron sends a pulse train of positive polarity and amplitudes $|u_+|/2$ to the first electrode of the memristor, with the number of pulses proportional to the value of x_i. Simultaneously, in case of $\delta_j > 0$ the j-th neuron sends a pulse train of negative polarity and amplitudes of $|u_+|/2$ to the second electrode of the memristor, with the number of pulses proportional to the value of $delta_j$, and the j-th neuron is inactive in case of $\delta_j < 0$.

In the second phase, the i-th neuron sends a pulse train of negative polarity and amplitudes $|u_-|/2$ to the first electrode of the memristor, with the number of pulses proportional to the value of x_i. Simultaneously, in case of $\delta_j < 0$ the j-th neuron sends to the second electrode of the memristor a pulse train of positive polarity and amplitudes of $|u_-|/2$, with the number of pulses proportional to the value of δ_j, and the j-th neuron is inactive in case of $\delta_j > 0$.

These two phases go sequentially one by one and form one full cycle of learning. It should be emphasized that both phases are always performed, irrespective of whether they have an effect or not. It is necessary that the neurons, located at the opposite ends of the memristor, work independently and rely only on the common schedule.

3. Experiments

All modeling experiments were performed in MATLAB. In experiments, a standard two-layer network with [784-110-10] architecture was trained on the MNIST data set [23]. The MNIST dataset consists of 70,000 handwritten digits out of which 60,000 are used for the training process and 10,000 for the testing process.

During the learning phase, we used minibatches of size 100 to speed up calculations. The initial weights were selected randomly from the uniform distribution, so that $w_{ij}^{(k)} \in [-a_k, a_k]$, where $a_k = 1/\sqrt{(n_{k-1})}$, n_{k-1} is the number of neurons in the $(k-1)$-th layer.

At the beginning, we used the neuron sigmoidal activation function of the following type:

$$x_i = f(z_i) = 1/(1 + e^{-z_i}) \tag{11}$$

In the further experiments, we used another neuron activation function called *relu* (rectified linear unit):

$$f(x) = \max\{0, x\} \tag{12}$$

The reason for the use of such activation function lies in the simplicity of its implementation in hardware. Despite the simplicity and the discontinuity of the derivative of (12) at $x=0$, this function is widely used in machine learning and yields excellent results. For the weights update we used the Eq. (6).

Three parameters, such as the method of multiplication, the matrix of error propagation, and the "continuality" or "discreteness" of variables in the implementation of Eq. (8), were varied. Each of three described parameters has two possible options. In particular:
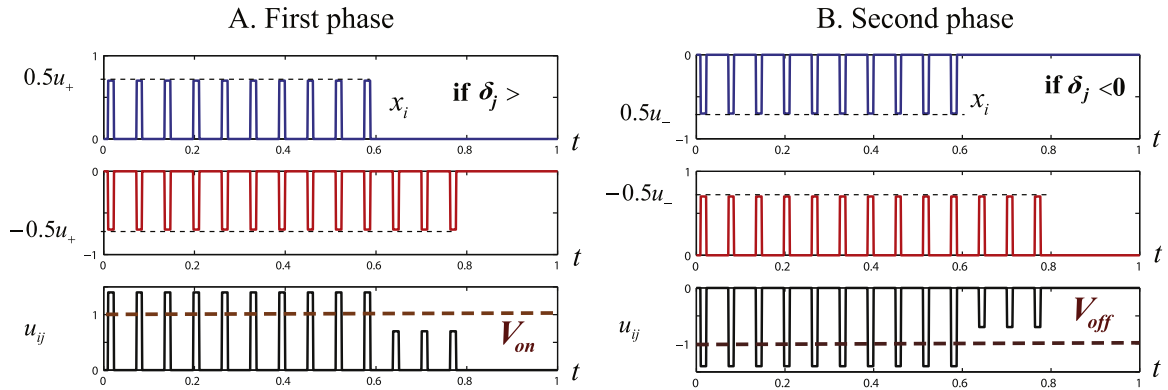

Fig. 4. Signal waveforms on memristor electrodes: the upper panel illustrates the pulse representation of the signal x_i at one electrode of a memristor, the middle panel illustrates the pulse representation of the signal δ_j at the opposite electrode of a memristor, and the bottom panel illustrates the potential difference u_{ij} across the memristor; (a) the waveform during the first phase resulting in the positive change of synaptic weight; (b) the waveform during the second phase resulting in the negative change of synaptic weight (time units, frequency and pulse width are arbitrary).
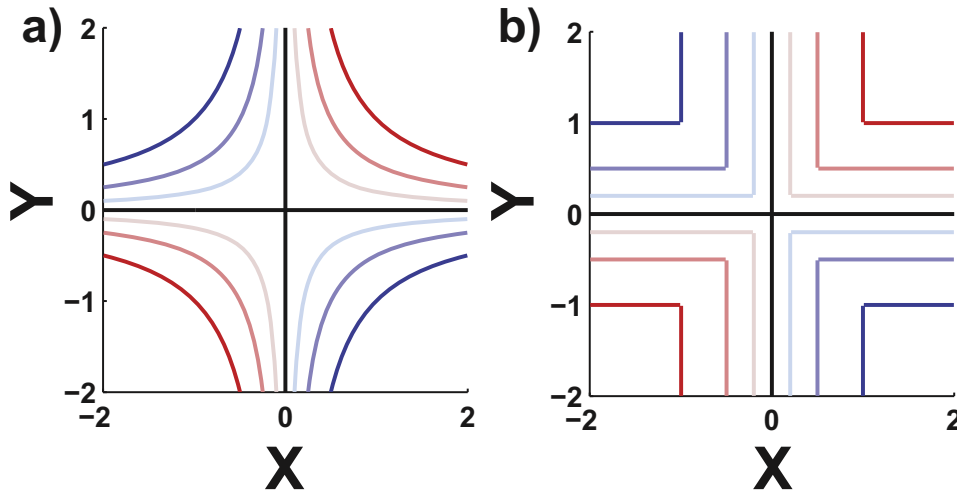
Fig. 5. The similarity between landscapes of values for a product function xy and $\mathrm{sign}(xy)\cdot\min(|x|, |y|)$ function; function value color notation increases from violet to blue and from yellow to red. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

1. As described in Section 2.3, two options of the multiplication method were examined. First of all, we used a regular mathematical multiplication procedure in accordance with (8). We refer to this procedure as the method *times*. The second option was to use the approximation (9), the method called *absmin*.
2. As described in Section 2.2, two options for the error propagation to the connection matrices were considered: (a) when the error is "back-propagated" across the same neuron connection weights, which works for the forward signal propagation (we called such standard method as *transposed*), and (b) when the error is conferred to the connection weights via the randomly chosen constant connection matrix, as has been specified in the Eq. (7) and in [21], which we called as *const* method.
3. Also, two cases in the implementation of (9) were considered: in the first one the values of x_i and δ_j are continuous, while in the second one these variables are made discrete before multiplication, as in the case illustrated in Fig. 4.

We assume that, when implemented in chip, the learning rate might depend on the frequency of pulses (see Fig. 4) and therefore it can be easily varied by changing the pulse generator frequency in the whole network. Subsequently, the operation of changing the learning rate can be easily achieved. In the modeling experiments, starting from the value of 10^{-4}, we dynamically change the learning rate depending on the training set error: if the error decreases, the epoch learning rate increases by 10%; if the error increases, the learning rate decreases by 30%. It is known that such dynamic change of the learning rate can speed up the learning process. We have found that in the experiment the learning rate almost always stays in the range from 10^{-4} to 10^{-3}.

As was described above, the values of x_i and δ_j are represented in the form of pulses, as shown in Fig. 4. However, in the modeling experiments, we have just discretized their values in the range from the experimentally found minimal to maximal values: $x_i \in [0, 5]$ and $\delta_j \in [-1, 2]$. The number of gradations can vary and in this paper we considered three cases: 20 and 100 gradations as well as a continuous one.

All the results were averaged by 10 trials.

4. Results

The results are shown in Fig. 6 and summarized in Table 1. In the simulation experiments, we used *relu* (12) activation function and dynamical adjustment of the learning rate.

In the experiment, the test error smoothly decreases to about 1.8%

for standard backpropagation and to 2.4% following the replacement of the *times* with *absmin* (black solid lines in Fig. 6).

In fact, the results obtained with the method *times* can be considered as standard and comparable with the state-of-the-art methods for this task. Significant improvements can be achieved only by increasing the number of hidden layer neurons and by increasing the training data set size with the use of elastic deformations of training examples [24].

The results obtained for x_i and δ_j discrete values are shown by dashed and dotted lines representing ≤100 and ≤20 pulses, respectively. One can see that given enough discrete levels (100 discrete values in our case) the performance is statistically close to that in a continuous case. However, the aggresive discretization leads to the rapid drop in the performance.

5. Discussion

The key issue addressed in our work is how to use the plasticity effects in synapses represented by memristors with multiple resistive states to locally implement the learning rule. The main distinction between our results and the related studies [25] is that we have implemented the mechanism, which is able to propagate error backwards and is needed for multi-layered networks. This is important for the deep learning schemes. Note that in [25] the single layer perceptron is considered, and the method proposed in that work cannot be used to propagate the error between the layers. The implementation of the learning rule requires the conversion of signals x_i and δ_j at opposite electrodes of a memristor in a crossbar to the voltage drop across the crossbar that would change the memristor conductivity proportional to the product $x_i \times \delta_j$. We propose the mechanism based on the pulsed representation of signals x_i and δ_j and implement the *absmin* operation instead of the product. The use of *relu* for the neuron transfer function also simplifies the implementation of neural networks as compared to the traditional sigmoidal transfer functions. Our results demonstrate that the memristor based implementation of error-based learning, including deep learning, is possible and can be efficient.

The issues arising due to the thermal noise and the variability in memristors are still not resolved, however, the proposed scheme appears to be quite robust with respect to the introduction of switching errors below certain threshold, because the learning procedure does not require setting exact values on each iteration of the process — the main requirement here is to move in the general direction of the error gradient. Hence, the reduction of the learning rate during such procedure should enable the system to eventually settle down in the
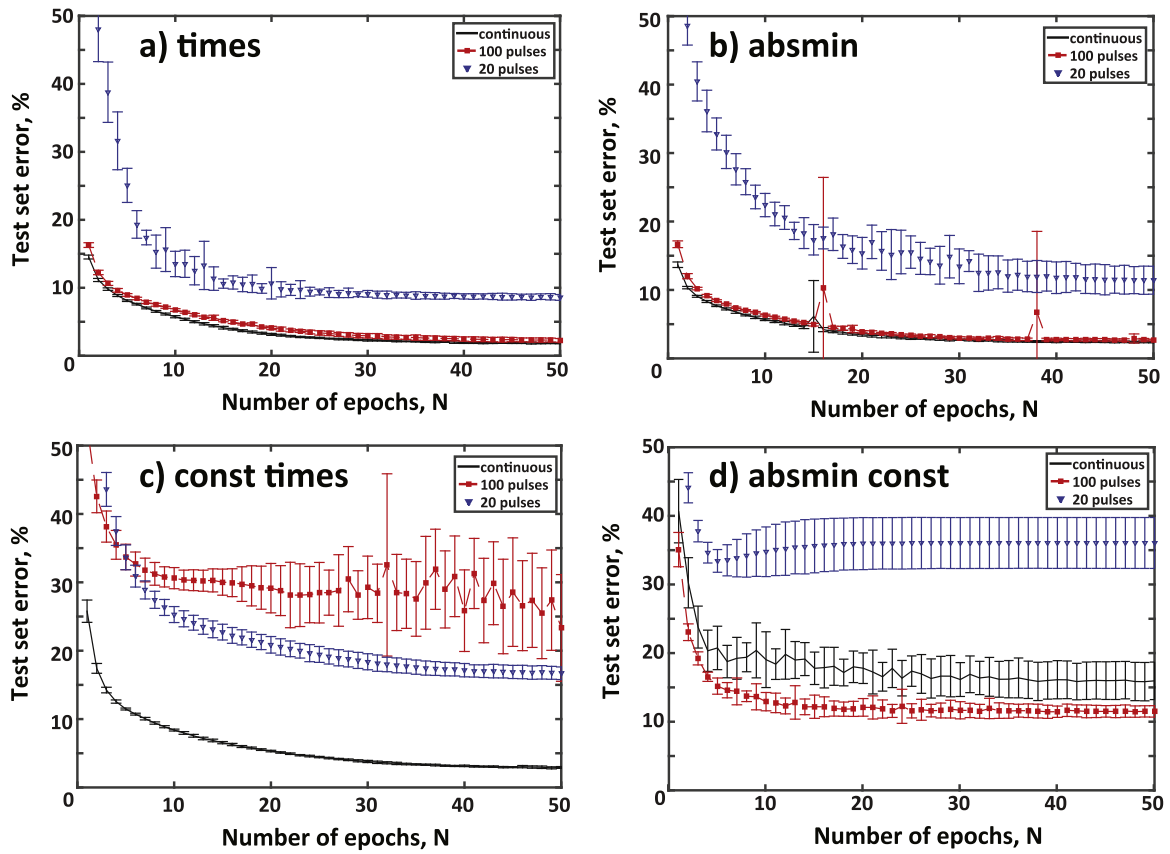
Fig. 6. Test set error for [784-110-10] network with relu-neurons and dynamically changing learning rate. The results for continuous values x_i and δ_j aswell as for the discreteness of 100 and 20 pulses are shown with black solid lines, red dashed-lines and the blue triangles, respectively: (a) method *times*, (b) method *absmin*, (c) method *times* with *constant* matrix, (d) method *absmin* with *constant* matrix. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 1
Test set errors for various methods, after 50 epochs of learning.

	Transposed		Const	
	Times	Absmin	Times	Absmin
Continuous	$1.8\% \pm 0.1\%$	$2.4\% \pm 0.1\%$	$2.9\% \pm 0.1\%$	$16.0\% \pm 2.7\%$
100 pulses	$2.3\% \pm 0.3\%$	$2.7\% \pm 0.1\%$	$23.3\% \pm 7.8\%$	$11.5\% \pm 0.8\%$
20 pulses	$8.6\% \pm 0.4\%$	$11.4\% \pm 2.0\%$	$16.7\% \pm 0.9\%$	$36.0\% \pm 3.7\%$

desired minimum. The verification of this behavior can be performed using Monte-Carlo simulations, but they require at least sufficient statistical data on element performance or a good physical model, which is not available at the present moment. At the same time, the initial studies of the effect of noise as well as parameter variability on the operation of memristor-based schemes demonstrate their significant tolerance to these factors [25]. This fact speaks in support of our arguments presented above.

The important issue arising during the implementation of the backpropagation learning rule is the routing of the error signal δ in backward direction. In this paper, we have considered two possible approaches: the propagation via the same connections (same memristors), which are used in the forward direction, or, alternatively, the propagation via a bypass connection matrix which adopts random values and remains constant during the learning (method *const*). Despite the simplicity of the implementation on chip of the latter approach, our experiments have shown unsatisfactory results using such method. Nevertheless, due to the symmetry of the memristor crossbar, the error backpropagation can be done in the same way as a forward propagation during the inference step by using current summation.

The proposed replacement of the *times* operation with *absmin* turns out to give good results. The modified learning rule generally does point into the similar direction (preserving the sign) as the original gradient and yields relatively smooth performance curves (Fig. 6b). However, it also have some shortcomings. Firstly, the gradient calculation becomes inexact, which results in slightly larger error (2.4% instead of 1.8%, see. Table 1). Secondly, the large price to pay is the necessity of conversion of x_i and δ_j into discrete pulses which seems laborious and decreases the accuracy for the small number of gradations.

On the other hand, in such pulse representation, the memristor conductance changes by small steps proportionally to the number of pulses. Our approach is very close to that proposed by [19], however, in our work we do not assume the linear dependence of the memristor conductance on the amplitude of the applied voltage. All pulses are assumed to be of the same amplitude, irrespective of the values of x_i and δ_j.

Finally, it should be emphasized that in this study we only demonstrate the operability of the proposed mechanisms at the proof of concept level. That is why we have restricted our work to the most commonly used benchmark— the MNIST database, and have applied the number of hidden units, which is sufficient to show that the proposed simplifications still provide learning operation quality, surpassing that of "shallow" perceptron. The detailed studies of models of memristor-based deep learning systems and their physical implementation will follow. We believe that the approaches proposed in our work can expand the capabilities of memristor technologies in the application for self-trained multilayer neural networks realization and decrease their power consumption by reducing the required number of memristors down to one-on-one connection weight. It is also worth noting that the techniques described in this paper are highly scalable.

Disclosure/conflict-of-interest statement

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Acknowledgments

References

[1] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (2015) 436–444.
[2] S. Adhikari, H. Kim, R. Budhathoki, C. Yang, L. Chua, A circuit-based learning architecture for multilayer neural networks with memristor bridge synapses, in: IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 62, 2015, pp. 215–223.
[3] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, J. Cong, Optimizing fpga-based accelerator design for deep convolutional neural networks, in: Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '15, ACM, New York, NY, USA, 2015, pp. 161–170. ⟨http://dx.doi.org/10.1145/2684746.2689060⟩.
[4] K. Likharev, Crossnets: neuromorphic hybrid cmos/nanoelectronic networks, Sci. Adv. Mater. 3 (2011) 322.
[5] B. Govoreanu, G.S. Kar, Y.Y. Chen, V. Paraschiv, S. Kubicek, F. et al., 10×10 nm^2 hf/hfo$_x$ crossbar resistive ram with excellent performance, reliability and low-energy operation., in: IEEE International Electron Devices Meeting (IEDM), 2011, pp. 31–36.
[6] G. Indiveri, L.-B. B., T. Hamilton, A. van Schaik, R. Etienne-Cummings, T. Delbruck, S.-C. Liu, P. Dudek, P. Hafliger, S. Renaud, J. Schemme, G. Cauwenberghs, J. Arthur, K. Hynna, F. Folowosele, S. Saighi, S. Serrano-Gotarredona, J. Wijekoon, Y. Wang, K. Boahen, Neuromorphic silicon neuron circuits., Sci. Adv. Mater., vol. 5, 2011, pp. 1–23.
[7] G. Indiveri, B. Linares-Barranco, R. Legenstein, T. Deligeorgis, G. Prodromakis, Integration of nanoscale memristor synapses in neuromorphic computing architectures, Nanotechnology 24 (2013) 384010.
[8] Y. Matveyev, K. Egorov, A. Markeev, A. Zenkevich, Resistive switching and synaptic properties of fully atomic layer deposition grown *tin/hfo$_2$/tin* devices, J. Appl. Phys. 117 (2015) 044901.
[9] L. Wang, H. Li, S. Duan, T. Huang, H. Wang, Pavlov associative memory in a memristive neural network and its circuit implementation, Neurocomputing 171 (2016) 23–29.
[10] X. Hu, G. Feng, S. Duan, L. Liu, Multilayer rtd-memristor-based cellular neural networks for color image processing, Neurocomputing 162 (2015) 150–162.
[11] L. Chua, Everything you wish to know about memristors but are afraid to ask, Radioengineering 24 (2015) 319–368.
[12] C. Merkel, D. Kudithipudi, R. Ptucha, Heterogeneous cmos/memristor hardware neural networks for real-time target classification., in: Proceedings of the SPIE 9119, Machine Intelligence and Bio-inspired Computation: Theory and Applications, vol. VIII, 2014, p. 911908.
[13] P. Werbos, Beyond Regression New Tools for Prediction and Analysis in the Behavioral Sciences, 1975.
[14] C. Merkel, D. Kudithipudi, Comparison of off-chip training methods for neuro-memristive systems., in: Proceedings of the 28th International Conference on VLSI Design (VLSID), 2015, pp. 99–104.
[15] F. Alibart, E. Zamanidoost, D.B. Strukov, Pattern classification by memristive crossbar circuits using ex situ and in situ training, Nat. Commun. 4 (2013) 2072.
[16] B. Widrow, C. Greenblatt, Y. Kim, D. Park, The no-prop algorithm: a new learning algorithm for multilayer neural networks, Neural Netw. 37 (2013) 182–188.
[17] K. Singh, C. Sahu, J. Singh, Linearly separable pattern classification using memristive crossbar circuits., in: Proceedings of the 15th International Symposium on Quality Electronic Design (ISQED), 2014, pp. 323–329.
[18] R. Hasan, T. Taha, Enabling back propagation training of memristor crossbar neuromorphic processors, in: Proceedings of the International Joint Conference on Neural Networks (IJCNN), 2014, pp. 21–28.
[19] D. Soudry, D. Castro, A. Gal, A. Kolodny, S. Kvatinsky, Memristor-based multilayer neural networks with online gradient descent training, IEEE Trans. Neural Netw. Learn. Syst. 26 (2015) 2408–2421.
[20] S. Jo, T. Chang, I. Ebong, B. Bhadviya, P. Mazumder, W. Lu, Nanoscale memristor device as synapse in neuromorphic systems, Nano Lett. 10 (2012) 1297–1301.
[21] T. Lillicrap, D. Cownden, D. Tweed, C. Akerman, Random feedback weights support learning in deep neural networks, arXiv, 2014. (1411.0247).
[22] A. Markeev, A. Chouprik, K. Egorov, Y. Lebedinskii, A. Zenkevich, O. Orlov, Multilevel resistive switching in ternary $hf_x al_{1-x} o_y$ oxide with graded al depth profile, Micro. Eng. 109 (2013) 342–345.
[23] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, in: Proceedings of the IEEE, vol. 86(11), 1998, pp. 2278–2324. ⟨http://dx.doi.org/10.1109/5.726791⟩.
[24] U. Meier, D. Ciresan, L. Gambardella, J. Schmidhuber, Better digit recognition with a committee of simple neural nets, in: Proceedings of the 2011 International Conference on Document Analysis and Recognition (ICDAR), 2011, pp. 1250–1254. http://dx.doi.org/10.1109/ICDAR.2011.252).
[25] P. Nair, M.V.and Dudek, Gradient-descent-based learning in memristive crossbar arrays., in: Proceedings of the 2015 IEEE International Joint Conference on Neural Networks (IJCNN), 2015, pp. C. 1–7.

Iakov Karandashev received the M.Sc. degree in applied physics and mathematics in 2010 from Moscow Institute of Physics and Technologies, Dolgoprudny, Russia, and the Ph.D. degree in combinatorial optimization from the Scientific Research Institute for System Analysis of Russian Academy of Sciences (SRISA RAS), Moscow, Russia, in 2013. Currently, he is a research assistant in Center of optical and neural technologies of SRISA RAS. His research interests include optimization, deep neural networks, and machine learning.
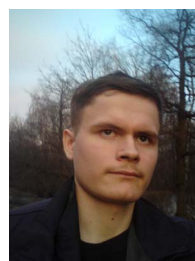
Vladimir Shakirov received the B.Sc. degree in applied physics and mathematics in 2011 from Moscow Institute of Physics and Technologies, Dolgoprudny, Russia. Currently, he is a research assistant in Center of optical and neural technologies of SRISA RAS. His research interests.

Yury Matveyev, Ph.D., senior researcher, laboratory "Functional materials and devices for nanoelectronics", Moscow Institute of Physics and Technology. Professional interests are synthesis of novel functional materials for nanoelectronics in ultrathin films and investigation of their chemical and electrical properties in multilayered structures.

Witali L. Dunin-Barkowski, D.Sc., Ph.D., Professor and Chair, Department of Neuroinformatics, Scientific Research Institute for System Analysis, Russian Academy of Sciences, Moscow. M.S. in Biophysics, Moscow Institute of Physics and Technology, 1965. President-Founder of the Russian Neural Network Society (Russian Association of Neuroinformatics, 1991). Professional interests in experimental and theoretical biophysics of neural systems, neurocomputing and artificial mind.

Andrei Zenkevich has graduated from Moscow Engineering Physics Institute (state university) in 1985 and received Ph.D. degree in 1997. He is currently the head of the laboratory "Functional materials and devices for nanoelectronics" in the Moscow Institute of Physics and Technology, Dolgoprudny, Russia. His research interests include the search for and investigation of physical mechanisms of storing information in the novel thin film materials/nanostructures, the functionalization of the memristor devices as electronic synapses and their integration in the prospective neuromorphic computing architectures.