A continuous-time learning rule for memristor-based recurrent neural networks

A Continuous-time Learning Rule for Memristor–based Recurrent Neural Networks

Gianluca Zoppo, Francesco Marrone, Fernando Corinto
Department of Electronics and Telecommunications
Politecnico di Torino
Torino, Italy
emails: gianluca.zoppo@polito.it; francesco.marrone@polito.it, fernando.corinto@polito.it

***Abstract*—Among the recent disruptive technologies, volatile/nonvolatile memory–resistor (memristor) has attracted the researchers' attention as a fundamental computation element. It has been experimentally shown that memristive elements can emulate synaptic dynamics and are even capable of supporting spike timing dependent plasticity (STDP), an important adaptation rule for competitive Hebbian learning. The overall goal of this work is to provide a novel analogue computing platform based on memristor devices and recurrent neural networks that exploit the memristor device physics to implement the backpropagation algorithm. Back propagation for recurrent neural networks requires a side network for the propagation of error derivatives. The use of memristor–based synaptic weights permit to propagate the error signals in the network via the nonlinear dynamics without the need of a digital side network. Experimental results show that the approach significantly outperforms conventional architectures used for pattern reconstruction. Further results will be reported in an extended work.**

I. Introduction

In the last few decades, the search of innovative computing platforms featuring an exponential market adoption has intensified, looking for technological areas that could offer new, ultra–low power processing methods and architectures. A neuromorphic computing approach aims to go beyond the status quo of conventional digital processing by exploiting complex dynamics and nonlinear phenomena emerging from the physics of nonvolatile memory devices (e.g. memristors) [1], [2]. Combining memristor technology with advanced deep learning algorithms used to train neural networks unlocks processing speed and power efficiency for large sets of sensor data. In supervised learning, one of the most popular method used for training feedforward neural networks is the backpropagation algorithm.

The aim of this paper is to propose a generalization of backpropagation to Recurrent Neural Networks (aka Recurrent Backpropagation). This method has been first introduced by Almeida [3] and Pineda [4] who independently obtained the same results and developed an iterative scheme to adjust the synaptic weight matrix of the neural network. The idea is to force the neural network to converge, for fixed input and initial state, to a desired fixed–point attractor. As for feedforward neural networks, this is achieved by minimizing a particular loss function associated to the neural network parameters. The novelty of this method is that the error signal is now "backpropagated" by introducing an analog side network (i.e. an associated differential equation). This avoids the direct computation of the gradient by exploiting the second nonlinear dynamics of the side network that adjusts the synaptic weights in situ. This work shows that the nonlinear dynamics of the analog side network can be mapped onto (the state equation) memristor–based synaptic weights. Although this work includes some selected experimental results, an extensive study has shown that the proposed approach outperforms conventional architectures used for pattern reconstruction. Further results will be reported in an extended work.

II. Memristor–based Recurrent Neural Network

Let each synaptic weight be described by a generic memristor (see also [5]) that satisfies the following equations:

$$\begin{cases} i = G(\mathbf{x})v \\ \frac{d\mathbf{x}}{dt} = f(\mathbf{x}, v) \end{cases} \tag{1}$$

where i is the current, v is the voltage, $G(\cdot)$ is the memductance and $\mathbf{x} = [w, y]^T$ is the state vector including the state variables w and y. By using a memristor–based synaptic weight $G(\mathbf{x}) = w$, the following set of ordinary differential equations describes the memristor–based neural network in Figure 1 ($\forall k = 1, \ldots, N$ and $\forall j = 1, \ldots, N$):

$$C_k \frac{dv_k}{dt} = -G_k v_k + G g_k \left(\sum_{j=1}^{N} w_{kj} v_j + I_k \right) \tag{2}$$

$$\frac{d\mathbf{x}_{kj}}{dt} = f(\mathbf{x}_{kj}, v_k) \tag{3}$$

In the next section a brief derivation of the backpropagation algorithm for recurrent neural network is reported in order to make clear the link between the nonlinear dynamics of memristor–based synaptic weights given by eq. (3) and the side network for the backpropagation of error signals. Hereinafter, $C_k = 1\,F$ and $G_k = G = 1\,\Omega^{-1}$ for the sake of simplicity.

III. Recurrent backpropagation algorithm

Consider a Recurrent Neural Network (RNN) whose state vector $\mathbf{v}$ evolves according to:

$$\frac{dv_i}{dt} = -v_i + g_i \left(\sum_{j=1}^{N} w_{ij} v_j + I_i \right), \qquad i = 1, \ldots, N \tag{4}$$

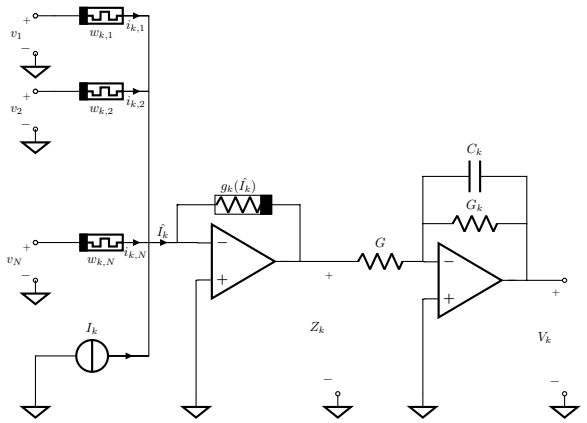

Fig. 1. Neuron's architecture with memristive synaptic connections.

where N is the number of neurons of the network and I_i is an external input to the i-th neuron. There is no restriction on the choice of the activation function g_i as long as it is differentiable [4]. In the most general case, it is possible to define different subsets of the network units:

- the subset *I* of input units;
- the subset *O* of output units;
- the subset *H* of hidden units.

The goal of the algorithm is to adjust the weights w_{ij} so that, for a given initial condition $\mathbf{v}^0 = \mathbf{v}(t_0)$ and a given vector of input $\mathbf{I}$, the RNN (4) converges to a desired fixed point $\mathbf{v}^\infty = \mathbf{v}(t_\infty)$. This is obtained by minimizing a loss function E which measures the euclidean distance between the desired fixed point and the actual fixed point:

$$E = \frac{1}{2}\sum_{i=1}^{N} J_i^2 = \frac{1}{2}\sum_{i=1}^{N}(T_i - v_i^\infty)^2 \tag{5}$$

where T_i is the i-th desired output state component and J_i is the i-th component of the difference between the current fixed point v_i^∞ and the target point T_i. Observe that E depends on the weight matrix $\mathbf{W}$ through the fixed point $\mathbf{v}^\infty(\mathbf{W}, \mathbf{I})$. Therefore, one way to drive the system to converge to a desired attractor is to let it evolve in the weight parameter space along trajectories which have opposite direction of the gradient of E:

$$\frac{dw_{ij}}{dt} = -\eta\frac{\partial E}{\partial w_{ij}}, \qquad \eta > 0 \tag{6}$$

where η is the learning rate and must be small enough so that the state variable $\mathbf{v}$ can always be considered to be at steady state [4]. Computing now the derivatives in (6), one obtains

$$\begin{aligned}\frac{dw_{ij}}{dt} &= -\eta\frac{\partial}{\partial w_{ij}}\left(\frac{1}{2}\sum_{k=1}^{N} J_k^2\right)\\ &= -\eta\sum_{k=1}^{N} J_k\frac{\partial J_k}{\partial w_{ij}} =\\ &= \eta\sum_{k=1}^{N} J_k\frac{\partial v_k^\infty}{\partial w_{ij}}\end{aligned} \tag{7}$$

and the derivative of v_k^∞ with respect to w_{ij} is derived by observing that the fixed points of (4) must satisfy the nonlinear equation:

$$v_k^\infty = g_k\left(\sum_{s=1}^{N} w_{ks}v_s^\infty + I_k\right). \tag{8}$$

Differentiating (8) with respect to w_{ij} one obtains:

$$\frac{\partial v_k^\infty}{\partial w_{ij}} = g_k'(\hat{I}_k^\infty)\left[\sum_{s=1}^{N}\frac{\partial w_{ks}}{\partial w_{ij}}v_s^\infty + \sum_{s=1}^{N} w_{ks}\frac{\partial v_s^\infty}{\partial w_{ij}}\right] \tag{9}$$

where $\hat{I}_k^\infty = \left(\sum_{s=1}^{N} w_{ks}v_s^\infty + I_k\right)$.

Solving (9) in terms of $\frac{\partial v_k^\infty}{\partial w_{ij}}$ and defining $L_{ks} = \delta_{ks} - g_k'(\hat{I}_k^\infty)w_{ks}$, it follows that:

$$\begin{cases}\sum_{s=1}^{N} L_{is}\frac{\partial v_s^\infty}{\partial w_{ij}} = g_i'(\hat{I}_i^\infty)v_j^\infty & k = i\\ \sum_{s=1}^{N} L_{ks}\frac{\partial v_s^\infty}{\partial w_{ij}} = 0 & k \neq i\end{cases} \tag{10}$$

and therefore for the generic k-th components

$$\frac{\partial v_k^\infty}{\partial w_{ij}} = L_{ki}^{-1}g_i'(\hat{I}_i^\infty)v_j^\infty \tag{11}$$

In conclusion, (6) simply becomes:

$$\frac{dw_{ij}}{dt} = \eta\left[g_i'(\hat{I}_i^\infty)\sum_{k=1}^{N} J_k L_{ki}^{-1}\right]v_j^\infty. \tag{12}$$

Unfortunately, (12) requires an inversion of L_{ki} for computing the weights' update but, considering

$$y_i = g_i'(\hat{I}_i^\infty)\sum_{k=1}^{N} J_k L_{ki}^{-1} \tag{13}$$

one can avoid this process by introducing an associated dynamical system. Indeed, assuming that $g_i'(\hat{I}_i^\infty) \neq 0\ \forall \hat{I}_i^\infty \in \mathbb{R}$ and observing that $L_{ki} = L_{ik}$ for construction, (13) is equivalent to:

$$\sum_{k=1}^{N} L_{ik}^{-1}J_k = \frac{y_i}{g_i'(\hat{I}_i^\infty)}. \tag{14}$$

Solving (14) in terms of J_k one obtains:

$$J_k = \sum_{i=1}^{N} L_{ki}\frac{y_i}{g_i'(\hat{I}_i^\infty)}. \tag{15}$$

Substituting now the explicit form for L_{ki}, (15) becomes:

$$0 = -y_k + g_k'(\hat{I}_k^\infty)\left(\sum_{i=1}^{N} w_{ik}y_i + J_k\right) \tag{16}$$

which can be seen as the steady state of the following side network:

$$\frac{dy_k}{dt} = -y_k + g_k'(\hat{I}_k^\infty)\left(\sum_{i=1}^{N} w_{ik}y_i + J_k\right). \tag{17}$$

Therefore, the system of differential equations is completely defined by:

$$\frac{dv_i}{dt} = -v_i + g_i\left(\sum_{j=1}^{N} w_{ij}v_j + I_i\right) \quad (18)$$

$$\frac{dy_k}{dt} = -y_k + g_k'(\hat{I}_k^{\infty})\left(\sum_{i=1}^{N} w_{ik}y_i + J_k\right) \quad (19)$$

$$\frac{dw_{ij}}{dt} = \eta y_i^{\infty} v_j^{\infty} \quad (20)$$

Observe that the weights' update depends on the corresponding fixed points of the first two equations, whereas the last two equations can be mapped onto the memristor state equation (3). It is readily derived that (18), (19), (20) describe a memristor–based recurrent neural network. Further details about the local stability of the system are reported in [3].

IV. Case study and simulations

In this section, a first investigation of training the memristor–based recurrent neural network (M–RNN) (18)–(20) by using the generalized backpropagation algorithm in a pattern's reconstruction task is presented. For this kind of application, input and output units coincide and no hidden units are considered, i.e. $I = O$ and $H = \emptyset$. Moreover, in order to guarantee the convergence of the system, symmetric weights were chosen for sake of simplicity. The dynamical system defined by (18), (19), (20) evolves in the following way:

- (18) evolves starting from a prefixed initial condition and converges to the corresponding fixed point $\mathbf{v}^{\infty}$;
- (19) evolves starting from a prefixed initial condition and converges to the corresponding fixed point $\mathbf{y}^{\infty}$;
- Lastly, the weights of the matrix $\mathbf{W}$ are adjusted accordingly to (20).

However, it is also important to mention that, using the correct relaxation time scales, the three steps listed above can be computed simultaneously. During the training phase, each image is repeatedly proposed to the network by mean of a constant input $\mathbf{I}$ until is memorized. In the case of multiple patterns to be learnt, the previous steps are performed for each single image of the dataset for different epochs. Observe that patterns were shown to the network in the same order for each epoch. This choice was not restrictive since similar performances were obtained even in the case the images were proposed in a random fashion. In order to train the network, the following parameters and initial conditions have been set for the training phase:

- Every time a pattern is shown to the network $\mathbf{v}(0) = (0.5, \ldots, 0.5)^T \in \mathbb{R}^N$ and $\mathbf{y}(0) = (0.5, \ldots, 0.5)^T \in \mathbb{R}^N$, as suggested in [4];
- The matrix $\mathbf{W}$ is symmetric and initialized with uniform random values between $[-0.1; 0.1]$;
- The activation function is chosen as a hyperbolic tangent function;
- The learning parameter $\eta = 0.01$ as suggested in [4];

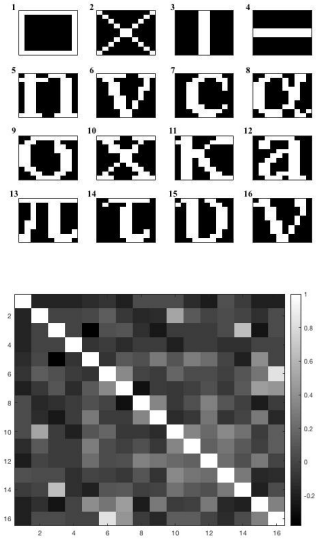

Fig. 2. In the top panel, the dataset of all the 16 patterns to be learnt. In the bottom panel, a graphical representation of the corresponding patterns' correlation matrix computed with the Pearson correlation coefficient.

- The number of epochs is 300.
- In order to guarantee the convergence of the three state variables $\mathbf{v}$, $\mathbf{y}$ and $\mathbf{W}$, adimensional time spans of $[0, 50]$, $[0, 20]$ and $[0, 3]$ were respectively used.

With the aim of discussing the feasibility of the M–RNN (18)–(20) in VLSI implementation, a short analysis on the importance of local/global interconnections is performed. Starting from a fully connected M–RNN with a full matrix $\mathbf{W}$ of synaptic weights, the number of connections is decreased by setting to zero all the elements of $\mathbf{W}$ located outside a band centered in the main diagonal. This optimization is studied because fully connected neural networks are really difficult to be implemented in analog circuitry. The cut of K outer diagonals from the matrix reduces the number of synapses from N^2 to $N^2 - K(K+1)$. In order to test the M–RNN, corrupted patterns are generated by flipping, with probability p, each pixel of an image from white to black and viceversa. In this pattern recognition analysis, a corrupted pattern is identified as reconstructed if the least square error with respect to the original images is less than 0.1. The results are shown in Fig. 3 with different levels of test images' corruption (e.g. $p = 0.1$, $p = 0.15$, $p = 0.2$ and $p = 0.25$). The assement is carried out by testing the recovery capabilities of the M–RNN against 1000 corrupted patterns for each class in Fig. 2.

The plot in Figure 3 reveals that, despite the presence of different levels of corruption, the accuracy is still higher than 90% even when a large amount of global connections are cut off.

Figure 5 shows an example of the reconstruction of 6 corrupted images. The performance of the M–RNN is remarkably stunning with an accuracy of 0.99. The noise is removed in almost all the test cases with only few exceptions.

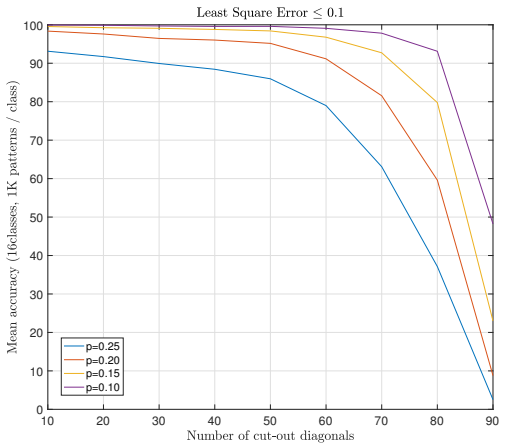

Fig. 3. Accuracy for different radius of connectivity with different levels of test images' corruption: $p = 0.10$, $p = 0.15$, $p = 0.20$ and $p = 0.25$.

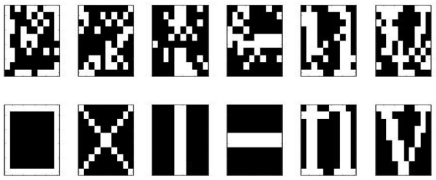

Fig. 4. On the top row, the corrupted patterns with probability $p = 0.2$. On the bottom row the corresponding reconstructed images with the fully connected network.

In order to assess the efficiency of the recurrent backpropagation algorithm, a comparison of this method is performed with two of the most used learning rules for training networks in associative memory's tasks. It is well known that a standard Hopfield model trained on uncorrelated patterns with the Hebbian rule has an approximate capacity of $0.14N$ (N is the number of units in the network). Unfortunately, this capacity decreases significantly if patterns are correlated. To overcome this problem, a novel learning method has been introduced by [6]. The Storkey learning rule presents indeed a significantly improved performance over the standard Hopfield model, both with correlated and uncorrelated data.

	Hebbian Rule	Storkey Rule	Pineda-Almeida Rule
Accuracy	0.1792	0.2663	0.9968

TABLE I
ACCURACY FOR EACH SINGLE LEARNING RULE OVER 1000×16 CORRUPTED IMAGES WITH PROBABILITY 0.1.

However, as shown in Table I and in the examples of Figure 5, the results provide compelling evidence that the recurrent backpropagation algorithm is able to reconstruct perfectly even in the presence of correlated patterns.

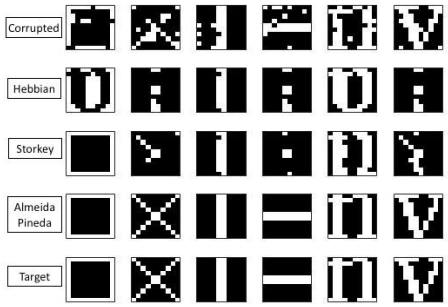

Fig. 5. From the top row: six corrupted patterns with probability $p = 0.1$, reconstructed pattern with hebbian rule, Storkey rule, Almeida-Pineda recurrent backpropagation rule and in the last row the target patterns.

V. CONCLUSIONS

In this paper, the dynamics of a memristor–based recurrent neural network has been analyzed. The network is trained by using a generalization of the recurrent backpropagation algorithm adapted to the continuous domain. Such in situ training learning rule permits to the memristor–based neural network to continuously adapt and adjust the synaptic weights without the direct computation of the loss function's gradient. The method significantly outperforms conventional approaches used for pattern reconstruction motivating future works for assessing the validity of the model in other areas of applications. However, further work is still needed to find physical devices that approximate the proposed memristive synapse dynamics. Nonetheless, the learning rule can be instead implemented by a series of discrete programming pulses that perform the weights update according to (20).

ACKNOWLEDGMENT

This work is supported by the Ministero degli Affari Esteri e della Cooperazione Internazionale (MAECI) under the project n. PGR00823. Authors are also grateful to Prof. Kyeong-Sik Min (School of Electrical Engineering, Kookmin Univ., Seoul, Korea) for the useful discussion.

REFERENCES

[1] L. O. Chua, "Memristor-the missing circuit element," IEEE Transactions on Circuits Theory, vol. 18, no. 5, pp. 507-519, 1971.
[2] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found" Nature, vol. 453, pp. 80-83, 2008
[3] L. Almeida, "A learning rule for asynchronous perceptrons with feedback in a combinatorial environment," Proceedings of the 1987 IEEE First Annual International Conference of Neural Networks, S. Diego, CA, 1987.
[4] F. Pineda, "Generalization of backpropagation to recurrent and higher order networks," Neural Information Processing Systems, D. Anderson (ed.), American Institute of Physics, 1987.
[5] F. Corinto, P.P. Civalleri, and L. O. Chua, "A theoretical approach to memristor devices," IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 2015.
[6] A.J. Storkey, "Increasing the capacity of the hopfield network without sacrificing functionality," International Conference on Artifcial Neural Networks, 1997.