Available online at www.sciencedirect.com
SCIENCE DIRECT®
Applied Soft Computing 4 (2004) 206–226
Applied Soft Computing
www.elsevier.com/locate/asoc

A dedicated very low power analog VLSI architecture for smart adaptive systems

Maurizio Valle*, Francesco Diotalevi[1]

Department of Biophysical and Electronic Engineering (DIBE), University of Genova, Via All'Opera Pia 11/A, I 16145 Genova, Italy

Received 15 January 2004; accepted 6 March 2004

Abstract

This paper deals with analog VLSI architectures addressed to the implementation of smart adaptive systems on silicon. In particular, we addressed the implementation of artificial neural networks with on-chip learning algorithms with the goal of efficiency in terms of scalability, modularity, computational density, real time operation and power consumption. We present the analog circuit architecture of a feed-forward network with on-chip weight perturbation learning in CMOS technology. Novelty of the approach lies in the circuit implementation of the feed-forward neural primitives and on the overall analog circuit architecture. The proposed circuits feature very low power consumption and robustness with respect to noise effects. We extensively tested the analog architecture with simulations at transistor level by using the netlist extracted from the physical design. The results compare favourably with those reported in the open literature. In particular, the architecture exhibits very high power efficiency and computational density and remarkable modularity and scalability features. The proposed approach is aimed to the implementation of embedded intelligent systems for ubiquitous computing.

Keywords: Smart adaptive systems; Analog VLSI neural networks; On-chip learning implementation; Perturbation based gradient descent algorithms; CMOS technology; Translinear circuits; Weak inversion; Differential and balanced current mode signal coding

1. Introduction

Intelligent/smart systems have become common practice in many engineering applications. This is mainly due to:

- the development along the last 30 years of powerful, robust and "adaptive" algorithms which can solve efficiently very computationally demanding tasks in many areas like robotics, decision making systems, artificial vision, pattern recognition, etc. We refer to the "soft computing" techniques like artificial neural networks, cellular networks, fuzzy logic, etc.;
- the exponential rate of advances in semiconductor technology, both in productivity and performance. This has allowed a tremendous ever-increasing trend in performance of microprocessors, digital signal processing devices, sensors solid-state interfaces, wireless systems, etc.;
- at last, an equivalent advance in silicon micromachining techniques and micro (nano) systems engineering.

Starting from the achieved results, the next breakthrough will be the design and development of "smart

* Corresponding author. Tel.: +39-010-353-2775; fax: +39-010-353-2096.
E-mail addresses: valle@dibe.unige.it (M. Valle), francesco.diotalevi@accent.it (F. Diotalevi).
[1] Present address: Accent Corporation Spa, Genova, Italy.

1568-4946/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2004.03.002

adaptive systems on silicon", i.e. very power and silicon area efficient devices which implement an entire system (i.e. sensing, computing and "actuating" actions) on a single silicon die. These systems should be able to "adapt" autonomously to the changing environment; they will not be programmed in a usual way but they will infer (operative) knowledge directly from the environment and from raw sensed data. In other words, they will be able to implement "intelligent" behaviour and "cognitive" tasks. In this paper we will concentrate on the VLSI implementation of cognitive systems (e.g. learning algorithms) and we will demonstrate that, by adopting weight perturbation (WP) learning algorithms and dedicated circuit design approaches, very efficient (in terms of scalability, modularity, computational density, real time operation and power consumption) analog and eventually mixed-mode implementations can be achieved.

It has been demonstrated elsewhere [1], that the computational density and the energy efficiency of dedicated analog on-chip learning chips greatly exceed those of their major competitors, i.e. state-of-the-art digital signal processors (DSPs). Moreover, the computational power of analog on-chip learning implementations is comparable to that of general-purpose microprocessors (see [2]) even if their computational density and energy efficiency is much greater. In fact the computational density and energy efficiency performance differ of many orders of magnitude in favour of dedicated analog neural chips even if they do not use state-of-the-art (and expensive) technologies. Even though circuit implemented with low cost technologies can achieve the above-mentioned performances, dedicated on-chip learning chips still exhibit high improvement capabilities. DSPs need A/D and D/A circuits to interface with sensors and actuators, thus, increasing the energy and silicon area budget. From this perspective, dedicated analog on-chip learning implementations can compete for speed and size with state-of-the-art DSPs. Their performances can be improved significantly if state-of-the-art technologies are used as well.

One should emphasize that the performances of the dedicated analog on-chip learning chips concur with those of analog computing arrays [3,4]. They exhibit a computing power in the order of 10^9 Operations Per S (OPS) per mW and per mm^2 and energy per operation (OP) in the order of 10^{-12} J. Analog on-chip learning implementations can be regarded as computing arrays with learning capabilities; they can take advantage of both of these features.

In this paper, we address the analog VLSI implementation of artificial neural networks with on-chip learning algorithms with the goal of efficiency in terms of scalability, modularity, computational density and power consumption.

We address scalability and modularity both at the system as well as at the circuit level. In the former case, we accomplish it by adopting weight perturbation gradient descent learning algorithms, which are more prone to scalable architectures. In the latter case, we adopt current-mode circuits; the current mode approach increases the dynamic range of signals and lowers significantly power consumption, if one implements circuits biased in weak inversion.

To increase the robustness of computation with respect to noise and analog circuit non-idealities, we adopt a differential and balanced current mode signalling approach and propose the on-chip learning implementation. In this perspective, one should observe that [1] the on-chip leaning scheme can successfully cope with the non-idealities and errors of the analog circuit implementation as the automatic on-chip tuning does with analog integrated filters. The feedback scheme is effective also because it exploits the speed of analog circuits.

Following the above considerations, we propose novel neural primitive circuits whose operation is based on the translinear approach: the circuits are biased in the weak inversion region of operation and achieve a very high power efficiency.

We present also the novel analog circuit architecture of a two-layer multi layer perceptron (MLP) network with an on-chip weight perturbation learning algorithm. The design has been done using a CMOS 0.8 μm technology (AMS CYE) [5], and considering a (low) power supply voltage of 2.5 V (i.e. negative power supply voltage $V_{SS} = -1.25$ V and positive power supply voltage $V_{DD} = +1.25$ V).

This paper is organised as follows. The system architecture (i.e. the learning algorithm and the overall analog circuit architecture) is detailed in Section 2. The rationale of the circuit design approach is presented in Section 3 while basic aspects of the design of the neural primitive and ancillary circuits are introduced in Section 4. The analog circuit architecture has

been extensively tested in learning simulations at circuit level using the netlist extracted from the physical design. The results and a comparison of the performances with the open literature are reported in Section 5. The conclusion is drawn in Section 6.

2. System architecture

2.1. Gradient descent learning algorithms

In the following, we will refer to the standard architecture of the feed-forward multi layer perceptron network without any loss of generality. In fact, in an MLP we may account for time by incorporating memory (e.g. tapped delay line) in the input layer of the network and use it for solving dynamic estimation problems. Alternatively, in either way, we may build feedback around an MLP by feeding the output signals of hidden layers back to the input of preceding layer(s). The application of the feedback configured in this manner not only turns a static MLP into a dynamic one (namely recurrent MLP) but also offers the potential for improving system performances [6].

In the standard architecture of a feed-forward N-layer MLP network, the lth layer ($l \in [1 \rightarrow N]$) is organised as a matrix of n^l neurons (where the suffix l stands for the layer number) connected to a matrix of $n^l \times n^{l-1}$ synaptic multipliers.

The neurons implement the linear summation of the synaptic outputs and the non-linear activation function $\Psi(\cdot)$. The computation performed by the jth neuron of the lth layer is:

$$a_j^l = \sum_{i=1}^{n^{l-1}} W_{j,i}^l X_i^{l-1} \qquad X_j^l = \tanh(a_j^l)$$
$$l \in [1 \rightarrow N],\ j \in [1 \rightarrow n^l],\ i \in [1 \rightarrow n^{l-1}] \tag{1}$$

where X_j^l is the output of the jth neuron of the lth layer, X_i^{l-1} the output of the ith neuron of the $(l-1)$th layer, the term a_j^l (usually called activation) is the sum of the outputs of the synapses of the $(l-1)$th layer which are connected to the jth neuron of the lth layer, $W_{j,i}^l$ is the weight of the synapse connecting the ith neuron of the $(l-1)$th layer to the jth neuron of the lth layer (the neuron layer 0, called the input layer, does not perform any computation but only feeds the network input signals X_i^0, $i \in [1 \rightarrow n^0]$). In the following, we will indicate the whole set of synaptic weights as $\boldsymbol{w}$. The synaptic weight values (together with the network topology) determine the network function, i.e. the non-linear mapping between input and output data spaces.

Supervised algorithms need training sets made of pairs of exemplary input patterns and desired output patterns (i.e. targets) [7,8]. The weights are updated until the target output is generated for the corresponding network input (i.e. an error index $\varepsilon(\boldsymbol{w})$ has been minimised, see below) or a pre-defined default termination condition is met (e.g. the number of iterations exceeds a given threshold). To accomplish this task, a gradient-descent algorithm is generally used.

When considering supervised "gradient descent" based learning algorithms, the learning task is accomplished by minimising, with respect to the adjustable parameters $\boldsymbol{w}$, the error index $\varepsilon(\boldsymbol{w})$; the update learning rule for the generic weight $W_{i,j}$ is:

$$\Delta w_{i,j} = -\eta \frac{\partial \varepsilon(\boldsymbol{w})}{\partial w_{i,j}} \tag{2}$$

where η is usually called "learning rate". The main issue from the analog VLSI point of view concerns the computation of $\partial\varepsilon(\boldsymbol{w})/\partial w_{i,j}$. By applying the chain rule, the back propagation (BP) algorithm calculates the value of $\partial\varepsilon(\boldsymbol{w})/\partial w_{i,j}$ while on the other hand the weight perturbation "estimates" it simply through its incremental ratio.

We define the error index ε_p for the pth pattern example as the sum (over all the output neurons) of the square difference between the target $\bar{X}_k^N$ and the actual output X_k^N (the index k runs over the output layer neurons):

$$\varepsilon_p(\boldsymbol{w}) = \frac{1}{2} \sum_{k=1}^{n^N} [\bar{X}_k^N - X_k^N]^2 \tag{3}$$

The error index $\varepsilon(\boldsymbol{w})$ is computed as the average of the error indices $\varepsilon_p(\boldsymbol{w})$ over all the N_p exemplary patterns of the training set:

$$\varepsilon(\boldsymbol{w}) = \frac{1}{N_p} \sum_{p=1}^{N_p} \varepsilon_p(\boldsymbol{w}) \tag{4}$$

The learning algorithm is operated by following an on-line or a by-epoch exemplary pattern presenta-

tion approach. When using the on-line approach, the patterns are sequentially (and usually in random order) given in input to the network. At each presentation, a weight contribution for each synapse is computed, and the synaptic weight values are updated. In the by-epoch approach, the weight update contributions are accumulated along each epoch while the weights are held constant. The synaptic weight values are then updated only once all the patterns have been presented to the network (i.e. the weight contributions related to all the pattern examples are computed).

With respect to the by-epoch approach, the on-line procedure introduces a kind of randomness in the learning process that may help in escaping from the local minima of the total error index $\varepsilon(\boldsymbol{w})$ [9]. Moreover, this technique is usually faster and more effective when the training set is huge (e.g. consisting of thousands of pattern examples as in the case of handwritten character recognition, speech recognition, etc.). On the other hand, the by-epoch approach usually gives better results when high "precision" is required in the non-linear mapping between input and output data spaces (e.g. function approximation) [1]. Nevertheless, the by-epoch approach requires more memory storage to accumulate the weight contributions computed at each pattern presentation [10]. The additional memory and the accumulation operation represent the main drawback of the implementation of the on-chip by-epoch learning in analog VLSI technology. In the following, we will focus on a by-pattern approach.

2.2. Weight perturbation learning algorithm(s)

The WP algorithms estimate rather than calculate the gradient's value of the output error index. This method is able to estimate the gradient simply through its incremental ratio: it measures the error index gradient by perturbing the network parameters (i.e. the weights) and by observing the change in the network output. If the weight perturbation $p_{j,i}^{(n)}$ at the nth iteration is small enough, we can neglect the higher order terms and write

$$\frac{\partial \varepsilon_p(\boldsymbol{w})}{\partial w_{j,i}} \cong \frac{\varepsilon_p(w_{j,i} + p_{j,i}^{(n)}) - \varepsilon_p(w_{j,i})}{p_{j,i}^{(n)}} \tag{5}$$

so

$$\Delta w_{j,i} = -\eta \frac{\varepsilon_p(w_{j,i} + p_{j,i}^{(n)}) - \varepsilon_p(w_{j,i})}{p_{j,i}^{(n)}} \tag{6}$$

where $p_{j,i}^{(n)}$ is the random perturbation injected in the $w_{j,i}$ synaptic weight at the nth iteration and $\Delta w_{j,i}$ is the value used to update the weight $w_{j,i}$. The difference between the error index before and after the perturbation of the generic weight $w_{j,i}$ is used to estimate the gradient's value with respect to $w_{j,i}$. This algorithm is the simplest form of the WP algorithm, and because only one synapse's weight is perturbed at a time, this technique is called sequential [11].

To make simpler the circuit implementation, every weight perturbation $p_{j,i}^{(n)}$ can be considered equal in value and random only in sign [12]:

$$p_{j,i}^{(n)} = \text{pert}_{j,i}^{(n)}\, \text{step}$$

where step is the perturbation value, while $\text{pert}_{j,i}^{(n)}$ can assume the values $+1$ or -1 with equal probability.

One can rewrite Eq. (6) as follows:

$$\begin{aligned}\Delta w_{j,i} &= -\eta \frac{\varepsilon_p(w_{j,i} + p_{j,i}^{(n)}) - \varepsilon_p(w_{j,i})}{p_{j,i}^{(n)}} \\ &= -\eta \frac{\Delta \varepsilon_p}{\text{step}} \text{pert}_{j,i}^{(n)} = -\frac{\eta}{\text{step}} \Delta \varepsilon_p \text{pert}_{j,i}^{(n)}\end{aligned} \tag{7}$$

The information of the term step in the η value can be combined, i.e.

$$\Delta w_{j,i} = -\eta' \Delta \varepsilon_p \text{pert}_{j,i}^{(n)}, \quad \eta' = \frac{\eta}{\text{step}},$$

$$\text{pert}_{j,i}^{(n)} = \begin{cases} +1 \\ -1 \end{cases} \quad \text{with equal probability} \tag{8}$$

To compute the synapse's weight update $\Delta w_{j,i}$, it is only necessary to compute $\Delta \varepsilon_p$ and to know $\text{pert}_{j,i}^{(n)}$. The box below shows the final version of the WP learning algorithm, in the case of a by-epoch pattern presentation approach.

The sequential WP algorithm may be too slow in real applications where big networks are employed. In [13] four parallel perturbation strategies have been highlighted. The one we will focus on is the "fully parallel weight perturbation" owing to its simpler circuit implementation with respect to the other perturbation strategies. Originally developed by Cauwenberghs

[14] and Alspector et al. [15], it is called stochastic error descent. It consists of the fully parallel perturbation of all synaptic weights. One can formulate the weight perturbation strategy as follows:

```
for(each epoch)
{set each pert_{j,i}^(0) at a random value;
        for(each pattern of the training set)
                {Choose a pattern in random way  and put it in input to the network;
                Feed-Forward phase;
                Compute ε_p(w_{j,i});
                Weight Perturbation;
                Feed-Forward phase;
                Compute ε_p (w_{j,i}+step·pert_{j,i}^(n));
                Compute Δw_{j,i}=−η'·Δε_p·pert_{j,i}^(n);
                WeightUpdate;
                }
}
```

```
For each pattern of the training set
    {apply perturbation to all weights;
    weight update;}
```

or in mathematical terms as: $\Delta \boldsymbol{w} = -\eta(\varepsilon_p(\boldsymbol{w}+\boldsymbol{p}^{(n)})-\varepsilon_p(w))\boldsymbol{p}'^{(n)}$, where $\boldsymbol{p}^{(n)}$ is the perturbation matrix of elements $p_{j,i}$ at the nth iteration and $\boldsymbol{p}'^{(n)}$ is a matrix of elements $1/p_{j,i}$ (inverse of perturbation) at the nth iteration.

2.3. Analog VLSI on-chip (supervised) learning implementation issues

The advantages of on-chip learning implementation are not evident when the problem solution requires small networks (i.e. with a small number of neurons and synapses) and the training set consists of few examples. In this case, the off-chip learning implementation is more convenient: a host computer can implement the learning algorithm by using a chip-in-the-loop technique [16]. Analog on-chip learning networks ought to be pursued when: (a) large networks (e.g. with thousands of synapses) and large training sets (e.g. thousands of examples) are considered; (b) we need to implement adaptive neural systems, i.e. systems that are continuously taught while being used.

As discussed in the previous section, multi-layer networks can be trained by using a supervised algorithm, i.e. the weight adjustment (see (2)) can be achieved by using the back propagation or the weight perturbation rules.

BP algorithms have been used extensively and successfully to solve real world tasks but they do not fully deal with the pitfalls of analog VLSI circuits. Therefore, one must take care into account when designing analog on-chip BP learning circuits. The BP algorithms compute the gradient of the error index function according to the transfer characteristics of synapses and neurons; nevertheless the behaviour of analog circuits (non-linearity effects, offsets, mismatches, technology process spread, etc.) affects the gradient computation and worsens the training performances.

The main drawbacks of the analog on-chip implementation of the BP algorithms are:

- fixed (in particular non-recursive) feed-forward network topology;
- need for circuits to compute the neuron function transfer derivative.
- sensitivity to circuit offsets;
- high signal count for the back-propagation of the error terms.

The WP rule looks more attractive for the analog on-chip implementation mainly because [1]:

- the WP algorithms can be applied also to recurrent multi-layer networks;
- the WP algorithms are more easily arranged in a scalable architecture;

- no need for circuits to compute the neuron function transfer derivative.
- the WP algorithms deal with non-idealities of analog VLSI circuits since no assumptions concerning the transfer function of neurons and synapses are taken into account. The estimation of the error function gradient does not depend on the linearity of the transfer function of neurons and synapses [11];
- lower sensitivity to circuit offsets with respect to BP algorithms;
- the weight update circuit is simpler if compared to the BP one (see for instance [17]). Weight updates (see (2)) are simply scalar multiples of the change in the index error [18];
- only one backward signal: $\Delta\varepsilon_p$, see (8);
- the signals (see below) that codify $\Delta\varepsilon_p$ are not managed by cascaded circuits (as in the back propagation implementation [10]) and then they are not further affected by incremental offset contributions.

Researchers in microelectronics designed WP algorithms with the aim to developing hardware friendly learning algorithms. Therefore, the researchers' activity concentrated on the algorithms and on their analog circuit implementation, and not on the performance in real applications with high computational burden. In previous papers (see for instance [19–23], we validated the WP learning algorithm in real applications and we found that BP offers slightly better performances (mainly in terms of learning iterations) if implemented on a general-purpose hardware platform. On the other hand, WP algorithms are much more suitable for the analog VLSI on-chip implementation. WP algorithms are much more reliable (and efficient) for the implementation of "cognitive" tasks in smart adaptive systems on silicon.

It has been demonstrated [18] that WP algorithms can achieve lower error rates than BP ones to train analog VLSI chips in the loop. Nevertheless, when using a generic hardware platform (i.e. personal computer or DPS acceleration board), the BP algorithms are more efficient with respect to WP ones: they usually achieve learning convergence in less iterations with comparable or lower classification performance.

2.4. Analog circuit architecture of a multi layer perceptron with on-chip weight perturbation learning algorithm

In this section, we will refer to the WP algorithm with the fully weight perturbation technique.

From (8), to compute the synaptic weight update term $\Delta W^l_{j,i}$, we only need to compute $\Delta\varepsilon_p$ and to known pert^l_{ji}. Then, we have to back-propagate only the following two signals:

- the absolute value of $\Delta\varepsilon_p$ (i.e. $|\Delta\varepsilon_p|$),
- the sign of $\Delta\varepsilon_p$ (i.e. sign $\Delta\varepsilon_p$).

The error index has been defined as follows:

$$\varepsilon_p = \sum_{i=1}^{n^2} |\bar{X}_i^2 - X_i^2| \tag{9}$$

i.e. the error function ε_p is equal to the sum of the absolute values of the differences between the MLP outputs and the target values.

Each synapse generates its own random perturbation sign (i.e. pert^l_{ji}). We do not address in this paper the random digital sign block generator circuit implementation because it has been successfully addressed elsewhere: two examples of circuits used to generate pseudorandom bit sequence are reported in [17,24]. The synaptic weights are on-line updated [9].

The on-chip learning algorithm [25], is detailed in Table 1 and it is also reported, for the sake of clarity, in Fig. 1.

Table 1
The on-chip learning algorithm

1	Select an input pattern out of the training set
2	Feed-forward phase
3	Output error computation and storage (i.e. $\varepsilon_p(\boldsymbol{w}) = \sum \lvert(\text{target output value} - \text{neuron output value})\rvert$)
4	Weights perturbation
5	Feed-forward phase
6	Perturbed output error computation and storage (i.e. $\varepsilon_p(\boldsymbol{w} + \boldsymbol{p}) = \sum \lvert(\text{target output value} - \text{neuron output value})\rvert$)
7	"Measurement" of the error gradient $\Delta\varepsilon_p$ see (8)
8	Back propagation to all synapses of the two signals: absolute value and sign of $\Delta\varepsilon_p$
9	Weights update: $\boldsymbol{w} = \boldsymbol{w} + \Delta\boldsymbol{w}$
10	If $\varepsilon_p(\boldsymbol{w})$ > given threshold goto point 1 elseif end

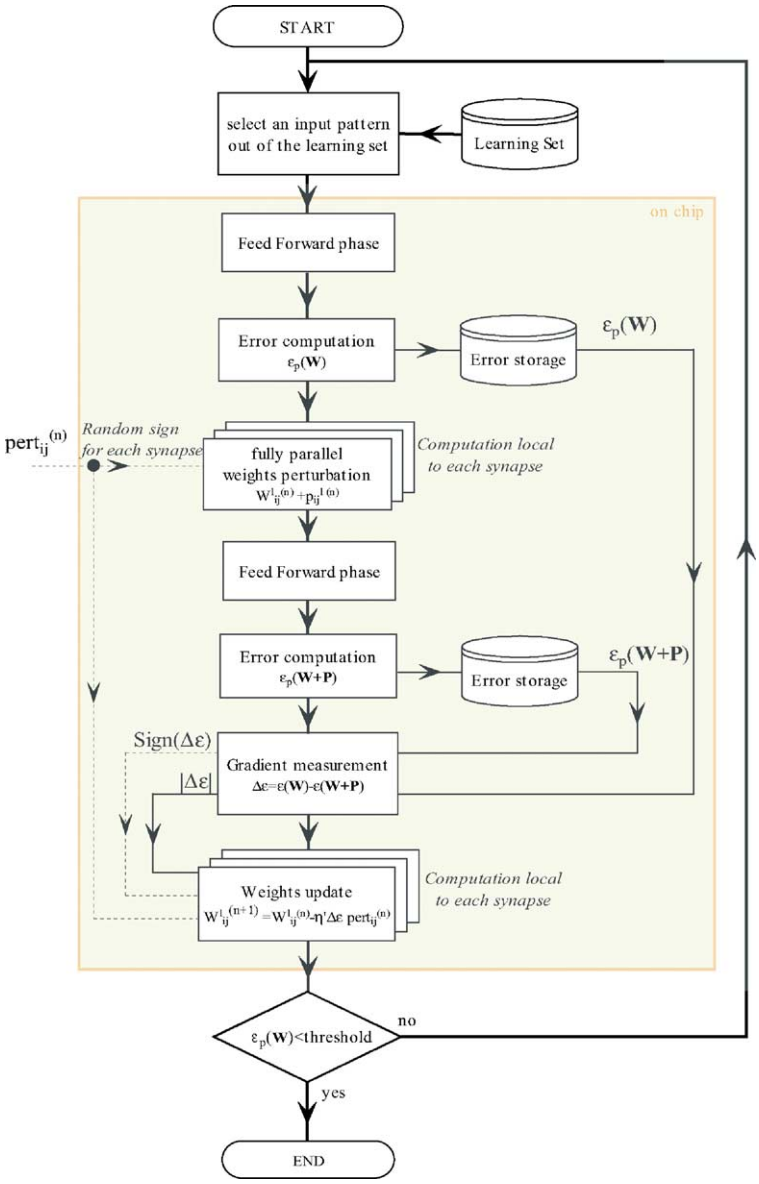

Fig. 1. The on-chip learning algorithm.

The procedure starts giving in input to the MLP a pattern out of the training set (point 1) and after the feed-forward phase (point 2) the output error (point 3) is computed and stored on-chip. All weights are simultaneously perturbed (point 4) and a new feed-forward phase (point 5) is performed to obtain the output of the MLP after the weights perturbation. The error index value (point 6) is stored on-chip as the previous one and the difference between them (point 7) (i.e. between the output errors after and before the weights perturbation) is computed. We obtain the error gradient value ($\Delta\varepsilon_p$) whose value is backward distributed (point 8) to all synapses in terms of: (1) a current signal (absolute value of $\Delta\varepsilon_p$); (2) a voltage signal (sign of $\Delta\varepsilon_p$). Based on these two signals and of the local information about the sign of the weight perturba-

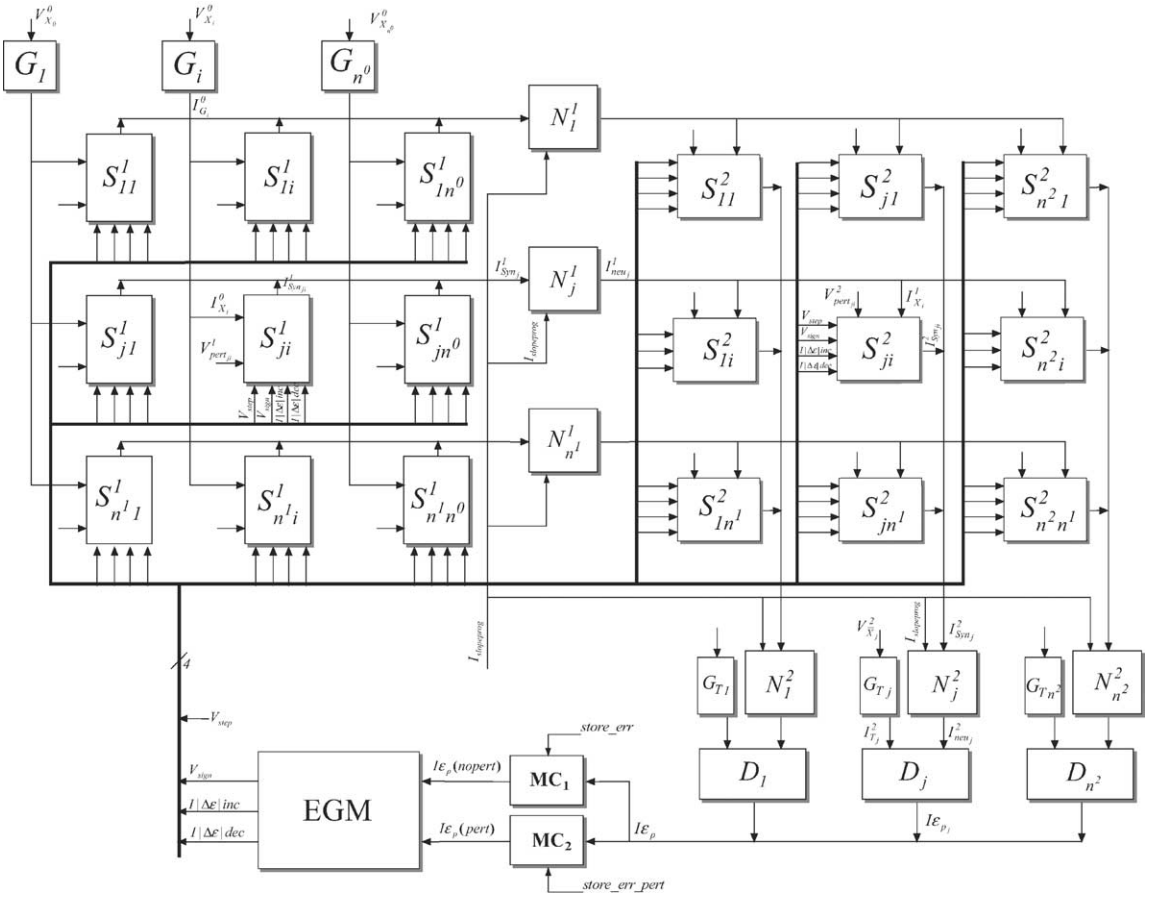

Fig. 2. Block diagram of the analog two layers MLP architecture with weight perturbation on-chip learning algorithm. Timing signals are not shown for clarity.

tion (i.e. pert_{ij}^{l}), we can proceed with the simultaneous computation of the weights update according to (8) (point 9). The external supervising system compares the error with a given threshold and it decides to continue or to stop the learning phase.

In Fig. 2, the block diagram of the analog two layers MLP architecture with the weight perturbation on-chip learning algorithm is shown. In Table 2, the correspondence between the neural and electrical variables has been detailed.

The variables of the algorithm need to be coded onto electrical signals such as voltages or currents. Normally, current signals are used for summing variables, by exploiting the Kirchoff current law (KCL), while voltage coding is used when a signal ought to be distributed to many modules. Table 3 lists the electrical coding used for neural signals in the circuit implementation.

3. Circuit design approach rationale

3.1. Current-mode circuits

Current-mode circuits exhibit high dynamic range even at low supply voltage values. Moreover, current-mode processing often leads to simpler circuitry and lower power consumption with respect to the voltage mode one [26]:

Table 2
Correspondences between the neural and electrical variables

Neural (algorithmic) variables	Electrical variables
X_i^{l-1}	$I_{G_i}^0$ $(l=1)$ I_{neu}^{l-1} $(l=2)$
W_{ji}^l	$I_{W_{ji}}^l$
$W_{ji}^l X_i^{l-1}$	$I_{\text{syn}_{ji}}^l$
a_j^l	$I_{\text{syn}_j}^l$
p_{ji}^l	$V_{p_{ji}}^l$
pert_{ji}^l	$V_{\text{pert}_{ji}}^l$
step	V_{step}
$\varepsilon_p(W)$	$I\varepsilon_p$(nopert)
$\varepsilon_p(W+p^{(n)})$	$I\varepsilon_p$(pert) $\Delta\varepsilon_p$, absolute value V_{sign}, sign
ΔW_{ji}^l	$I_{\Delta W_{ji}}^l$ (depending on $I\lvert\Delta\varepsilon\rvert$inc or $I\lvert\Delta\varepsilon\rvert$dec)
η'	Time duration of *Enable_update*
$\bar{X}_i^2$	$I_{\text{T}_j}^2$
$\lvert\bar{X}_i^2 - X_i^2\rvert$	$I\varepsilon_{p_i}$
ε_p	$I\varepsilon_p$

- a current mirror allows, in a very simply way, to distribute current signals to many other circuits;
- using translinear circuits (see below), the multiplication can be implemented in a very simple but effective way;
- the adder circuit is simply a node (i.e. by exploiting the Kirchoff current law).

The current mode approach is an efficient approach in neural network circuit design thanks to its inherent simplicity.

3.2. The translinear circuit design approach

The *translinear principle* (TP) [27] exploits the exponential current–voltage characteristics in bipolar transistors and offers a powerful circuit analysis and synthesis [28] tool. Originally formulated for bipolar transistors, this principle enables the design of analog circuits that perform complex computations in the current-domain including products, quotients, and power terms with fixed exponents. In analog neuromorphic VLSI systems [29–31], translinear circuits using MOS transistors biased in the weak inversion region have been successfully demonstrated.

Table 3
Signals coding summary list

Signals	Voltage	Current	Digital	Analog	Differential	Single ended	Learning	Feed-forward
$V_{X_i}^0$	✓			✓		✓		✓
$I_{G_i}^0$		✓		✓	✓			✓
$V_{\bar{X}_j}^2$	✓			✓		✓		✓
$I_{\text{T}_j}^2$		✓		✓		✓		✓
I_{neu}^l		✓		✓	✓			✓
$I_{\text{Slope Prog}}$		✓		✓		✓		✓
$V_{W_{ji}}^l$	✓			✓		✓		✓
$I_{W_{ji}}^l$		✓		✓	✓			✓
$I\lvert\Delta\varepsilon\rvert$inc, $I\lvert\Delta\varepsilon\rvert$dec		✓		✓		✓	✓	
$I_{\text{syn}_{ji}}^l$		✓		✓	✓			✓
$I_{\text{syn}_j}^l$		✓		✓	✓			✓
$V_{p_{ji}}^l$	✓			✓		✓	✓	
$V_{\text{pert}_{ji}}^l$	✓		✓			✓	✓	
V_{step}	✓			✓		✓	✓	
$I\varepsilon_p$(nopert)		✓		✓		✓	✓	
$I\varepsilon_p$(pert)		✓		✓		✓	✓	
$I\lvert\Delta\varepsilon\rvert$		✓		✓		✓	✓	
V_{sign}	✓		✓			✓	✓	
$I\varepsilon_{p_i}$		✓		✓		✓	✓	
$I\varepsilon_p$		✓		✓		✓	✓	

3.3. Information coding through differential balanced signals

To increase the robustness of the analog subsystem with respect to the noise injected through the substrate and/or the power supply lines by the digital subsystem, the signals of the neural architecture have been coded by differential and balanced current signals. Through differential balanced signals, one can:

- increase linearity trough the cancellation of the even order harmonics;
- lower the common mode noise effects (see above).

A generic balanced differential signal S is defined as the difference of two components S^+ and S^- as: $S = S^+ - S^- = xS_0$, where: $S^+ = 1/2S_0(1+x)$ and $S^- = 1/2S_0(1-x)$. S_0 is the bias term and x is the information-carrying signal. The x signal modulates the common bias signal S_0.

In the proposed architecture, S_0 corresponds to the bias current $I_B = 250\,\text{nA}$ and the information carrying variable (e.g. x in the previous example) varies in the range $[-1, +1]$: S^- and S^+ vary in the range [0, +250 nA], S in the range [−250 nA, +250 nA].

3.4. Signals distribution and normalisation

Current mirrors distribute in a simple way current signals. The sum of the synaptic outputs feed each neuron. The outputs of the synaptic blocks are differential balanced current signals then it is sufficient to connect them to a single node to sum them (by exploiting the KCL). To increase the modularity of the architecture, the sum of the synaptic output currents (i.e. I_{syn} in the following) in input to a given neuron is normalised with respect to the overall number of synapses connected to that neuron. Each single component of the synaptic differential output currents (I^+_{syn} and I^-_{syn}), owing to the current coding scheme, is normalised separately by using current mirrors. In principle this implementation is easy scalable and it is not dependent, to a first extent, on the number of synapses connected to each neuron. If N is the number of synapses connected to a single neuron, the neuron input currents are expressed by:

$$I^+_{neu} = \frac{1}{N}\sum_{i=1}^{N} I^+_{syn_i} \quad \text{and} \quad I^-_{neu} = \frac{1}{N}\sum_{i=1}^{N} I^-_{syn_i}$$

3.5. Precision considerations

A major concern is the poor matching performance of MOS transistors in weak inversion [30]. This is more severe in analog VLSI systems, where small geometry transistors are used to achieve high silicon area efficiency. In fact low power design considerations suggest that it is preferable to operate the devices in the region where the transconductance per unit current is the highest, i.e. in the weak inversion. Small geometry devices and high transconductance per unit current makes the drain current strongly dependent on spatial variations of technological parameters.

The benefit of an on-chip learning architecture on the ultimate system performance is evident in the design of the network. The feedback properties inherent in the learning architecture mitigate the intrinsic random variations in the device characteristics leading to a robust performance. The on-chip learning algorithm allows the analog VLSI implementations using poorly matched, small geometry and nano-power devices available in the CMOS technology.

4. Neural primitive and ancillary circuits

In the following, the blocks constituting the architecture shown in Fig. 2 are briefly introduced and hints about the CMOS circuit implementation are given.

4.1. The input transconductor block, G

The input transconductor block G_i, codifies the single-ended voltage inputs into the differential balanced currents. The design of the input transconductor circuit derives from the fully balanced transconductor amplifier with a four-transistors input stage [32]. The transconductor output current signal can be expressed as

$$I^0_x = I^{0+}_x - I^{0-}_x = 2g_m V_{in}$$

where g_m is the transconductance of the transistors forming the input differential pair, and $2V_{in}$ is the differential voltage input ($V^+_{in} = |V^-_{in}| = V_{in}$).

If we define the synaptic input value as x and if we set: $I^{0+}_G \equiv (1+x)I_B/2$ and $I^{0-}_G \equiv (1-x)I_B/2$ where I_B is a constant bias current value, we obtain: $I_x =$

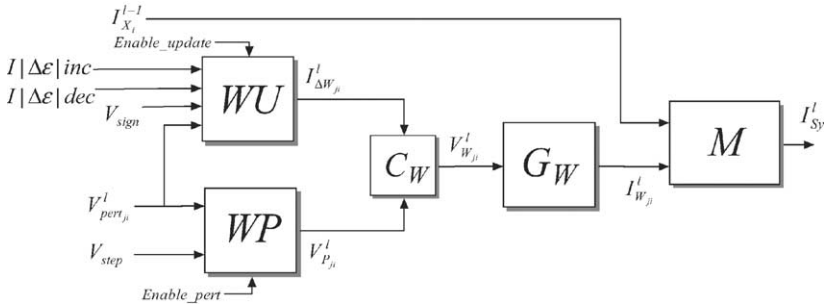

Fig. 3. Schema of the synapse (block S_{ji}^{l}).

$I_x^{0+} - I_x^{0-} = xI_B$ where $x = 2g_m V_{in}/I_B$. Please, note that $-1 \leq x \leq 1$. To obtain the two differential input voltages V_{in}^{+} and V_{in}^{-} a "single ended to differential" voltage-stage is used.

The simulated cut-off frequency of the input transconductor block G is of about 1 MHz. The simulated total harmonic distortion of the output current I_G obtained by an input sinusoidal voltage signal with 200 mV peak to peak results: 0.8% at 10 kHz, 1.5% at 100 kHz, and 3.2% at 1 MHz.

4.2. The synapse block, S

Fig. 3 shows the block diagram of the generic synapse block.

The weight voltage value $V_{W_{ji}}^{l}$ is dynamically stored in the block C_W. The block G_W translates $V_{W_{ji}}^{l}$ into the differential balanced current $I_{W_{ji}}^{l}$. The block M (synaptic multiplier) computes the product $I_{W_{ji}}^{l} I_{X_i}^{l-1}$. The block WP performs the perturbation of the weight voltage stored on C_W: depending on V_{step} and $V_{pert_{ji}}^{l}$, it temporally perturbs the voltage $V_{W_{ji}}^{l}$ by the quantity $V_{P_{ji}}^{l}$. The block WU performs the update of the weight voltage stored in C_W: depending on V_{sign}, and $V_{P_{ji}}^{l}$ it updates the voltage $V_{W_{ji}}^{l}$ feeding the block C_W with the quantity $I_{\Delta W_{ji}}^{l}$. The value of $I_{\Delta W_{ji}}^{l}$ can be equal to:

- $I|\Delta\varepsilon|$inc, in case of positive update;
- $I|\Delta\varepsilon|$dec, in case of negative update.

The *Enable_update* and *Enable_pert* signals control the time duration of the weight update and of the perturbation phases.

4.2.1. The weight transconductor, block GW

Following the same considerations done for the input transconductor (block G_i), we designed the weight transconductor (i.e. the transconductor that translates the weight voltage stored onto a capacitor into a differential balanced current I_W). The transconductor output current I_W is:

$$I_W = I_W^{+} - I_W^{-} = 2g_m V_W$$

where g_m is the transconductance of the transistors forming the input differential pair, and $2V_W$ is the differential weight voltage input ($V_W^{+} = |V_W^{-}| = V_W$).

The synaptic weight value is w and if we set

$$I_W^{+} \equiv \tfrac{1}{2}(1+w)I_B \quad \text{and} \quad I_W^{-} \equiv \tfrac{1}{2}(1-w)I_B$$

where I_B is a constant current value we obtain: $I_W = I_W^{+} - I_W^{-} = wI_B$ where $w = 2g_{mM3,M6}V_W/I_B$. Please, note that $-1 \leq w \leq 1$.

4.2.2. The weight update block, block WU

The weight update block (block WU) updates the weights stored in each synapse. Fig. 4 shows the WU circuit.

The signals $I|\Delta\varepsilon|$inc and $I|\Delta\varepsilon|$dec are equal to the absolute value of the error gradient $|\Delta\varepsilon_p|$ (signal of Table 1).

During the feed-forward phase or the weight perturbation phase, the switches S1 and S2 are open and the voltage V_W is stored on C_W. Following (8), when the signal *Enable_update* goes "high", the logic gates block (depending on the sign of $\Delta\varepsilon_p$ and of the random perturbation sign $V_{pert_{ji}}^{l}$, local to each synapse), manages the weight update.

As shown in Table 4, the truth table of the logic gates block of Fig. 4 is the following:

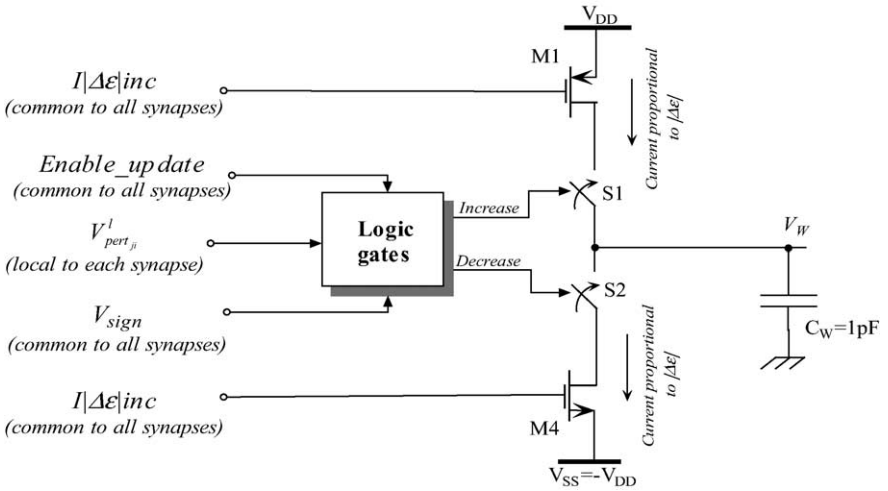

Fig. 4. Circuit schema of the weight update block (block WU).

- when *Enable_update* is "low", no weight update is performed;
- when *Enable_update* is "high" and the signals $V^l_{\text{pert}_{ji}}$ and V_{sign} have the same values (i.e. respectively, the sign of the weight perturbation and the sign of the error gradient are equal) the V_W voltage value is updated (i.e. decreased by the amount $|\Delta V_W|$);
- when *Enable_update* is "high" and the signals $V^l_{\text{pert}_{ji}}$ and V_{sign} have different values (i.e. respectively, the sign of the weight perturbation and the sign of the error gradient are different) the V_W voltage value is updated (i.e. increased by the amount $|\Delta V_W|$).

The value of the weight voltage update $|\Delta V_W|$ depends on the current $I|\Delta\varepsilon|$ that codifies the error gradient (see Table 1) and by the time duration Δt of the *Enable_update* signal (i.e. η' of (8)) as follows:

$$|\Delta V_W| = \frac{\Delta t I|\Delta\varepsilon|}{C_w}$$

Table 4
Truth table of the logic gates block (see Fig. 4)

Inputs			Outputs		Note
Enable_update	$V^l_{\text{pert}_{ji}}$	V_{sign}	Increase	Decrease	
0	*X*	*X*	1	0	No weight update
1	0	0	1	1	$\Delta V_W < 0$
1	0	1	0	0	$\Delta V_W > 0$
1	1	0	0	0	$\Delta V_W > 0$
1	1	1	1	1	$\Delta V_W < 0$

4.2.3. The weight perturbation block, block WP

The weight perturbation block (block WP) perturbs the weight stored in each synapse during learning. Fig. 5 shows the WP circuit schema. During the feed-forward phase or during the update of the weight voltage V_W, the switch S2 is closed while the switches S1 and S3 are open; the weight voltage V_W is translated by the weight transconductor G_W into a differential balanced current.

To perform a weight perturbation (i.e. *Enable_pert* = "*high*") the switch S2 is open, while one out of the switches S1 and S3 is closed depending on the value of the signal $V^l_{\text{pert}_{ji}}$ (local to each synapses).

We have set $C_W = 1\,\text{pF}$ and $C = 0.1\,\text{pF}$. In this way if *pert* = "*low*", the voltage V_W decreases by the quantity:

$$\Delta V^-_{W_{\text{pert}}} = |V^-|\frac{C}{C_W + C} = \frac{|V^-|}{11} \tag{10}$$

Vice versa, when *pert* = "*high*" the voltage V_W increases by the quantity:

$$\Delta V^+_{W_{\text{pert}}} = |V^+|\frac{C}{C_W + C} = \frac{|V^+|}{11} \tag{11}$$

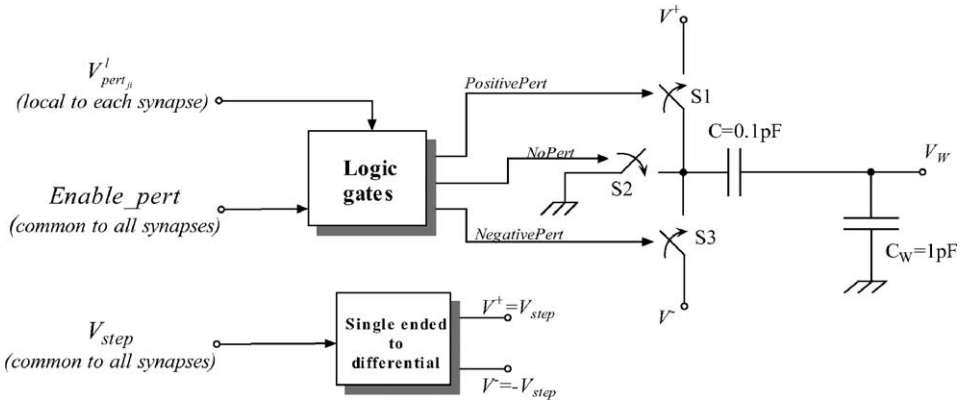

Fig. 5. Circuit schema of the weight perturbation block.

Setting $|V^+| = |V^-| = V_{\text{step}}$ thanks to a single ended to differential module, we can consider the absolute value of the amplitude perturbation $\Delta V_{W_{\text{pert}}}$ equal either for positive and negative perturbations. During the presentation of the generic kth pattern, for positive or negative perturbations, the amplitude of the weight perturbation is equal to $V_{\text{step}}/11$ as shown by (10) and (11). In other words:

$$\Delta V_{W_{\text{pert}}} = \frac{V_{\text{step}}}{11}$$

- if pert = "low" ⇒ negative perturbation, i.e. $V^k_{W\,\text{perturbed}} = V^k_W - \Delta V_{W\,\text{pert}}$;
- if pert = "high" ⇒ positive perturbation, i.e. $V^k_{W\,\text{perturbed}} = V^k_W + \Delta V_{W\,\text{pert}}$;

where k is the iteration index.

After the parallel weight voltages perturbation (see point 4 of Table 1), we measure the output error of the perturbed network (see point 6 of Table 1). Then we store this value and we restore the weights voltage values as they were before the weights perturbation; then we perform the weights update (see point 9 of Table 1).

The restore phase is performed by closing the switch S2 and:

- opening S1, if the perturbation is positive;
- opening S3 if the perturbation is negative.

The truth table of the logic gates block shown in Fig. 5 is reported in Table 5.

The signal V_{step}, used to set the voltage perturbation, is an external voltage common to all synapses as the signal *Enable_pert*, while the signal $V^l_{\text{pert}_{ji}}$, the random perturbation signal, is local to each synapse.

4.2.4. The synaptic multiplier M

The four quadrant synaptic multiplier ([33,34], see Fig. 6) is based on the translinear principle. It multiplies the input x (x varies in the range $[-1, +1]$) times the weight value w (w varies in the range $[-1, +1]$) as follows:

$$I_{\text{syn}} = xwI_{\text{B}}$$

where I_{syn} is the output synaptic current. Both the input and the weight values are coded by differential and balanced current mode signals:

$$I^+_x = \tfrac{1}{2}(1+x)I_{\text{B}}, \qquad I^-_x = \tfrac{1}{2}(1-x)I_{\text{B}}$$

$$I^+_w = \tfrac{1}{2}(1+w)I_{\text{B}}, \qquad I^-_w = \tfrac{1}{2}(1-w)I_{\text{B}}$$

The meaning of the circuit variables is the following:

Table 5
Truth table of logic gates of the WP block (see Fig. 5)

Inputs		Outputs			
Enable_pert	$V^l_{\text{pert}_{ji}}$	No pert	Positive pert	Negative pert	
0	X	1	1	0	No perturbation
1	0	0	1	1	Negative perturbation
1	1	0	0	0	Positive perturbation

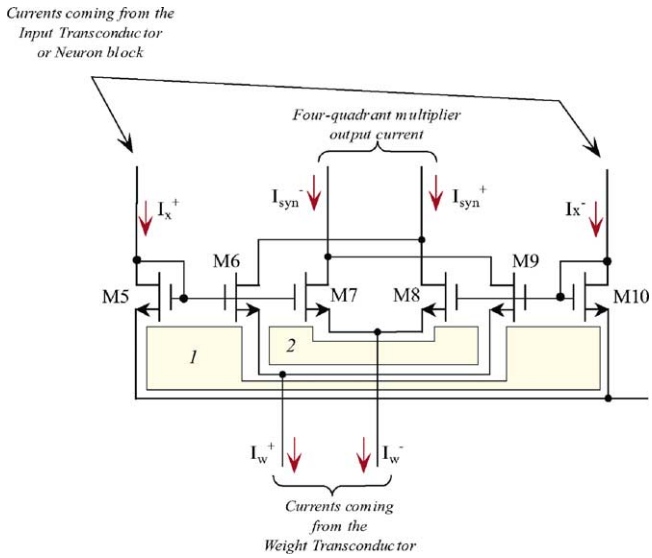

Fig. 6. Basic circuit schema of the four-quadrant translinear multiplier (block *M*).

- I_x^+ and I_x^- are the positive and the negative input current components;
- I_w^+ and I_w^- are the positive and the negative weight current components;
- I_{syn}^+ and I_{syn}^- are the positive and the negative output current components.

I_x^+, I_x^- and I_w^+ and I_w^- are managed in input to the multiplier by current mirrors as well as I_{syn}^+ and I_{syn}^- at the output.

Fig. 7 shows the dc simulated transfer characteristic of the multiplier. On the *x*-axis the input *x* is reported and the curves are parameterised with respect to the weight *w*.

Both the positive and negative current components of I_{syn} vary between 0 and 250 nA, while I_{syn} varies between −250 and 250 nA. The simulated −3 dB bandwidth is of about 1 MHz. The simulated total harmonic distortion of the multiplier where 250 nA dc current and 100 kHz 200 nA I_{pp} sinusoidal waves are applied

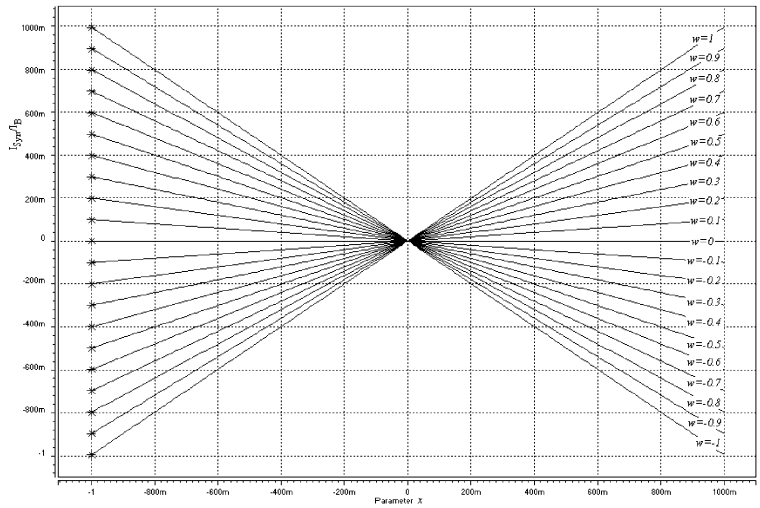

Fig. 7. Simulated dc characteristics of four-quadrant *M* multiplier as function of the input *x* with *w* in the range [−1, 1].

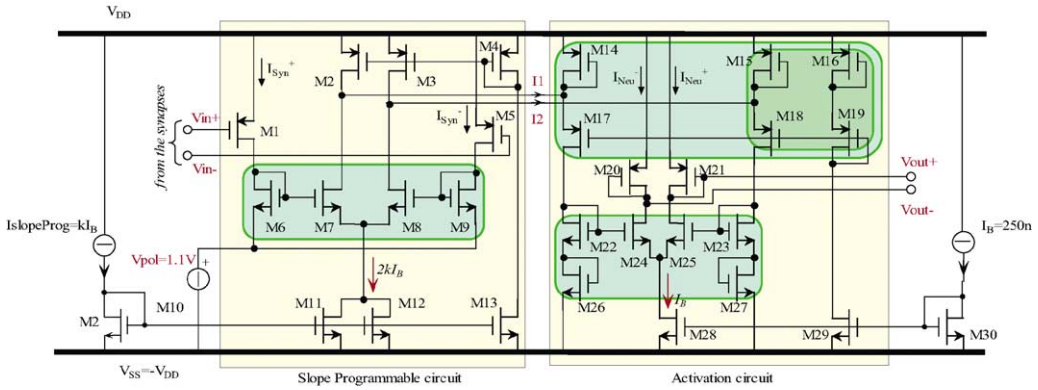

Fig. 8. Circuit schema of the neuron block *N*.

to, respectively, I_x (where $I_x = I_x^+ - I_x^- = xI_B$) and I_W (where) is 3% (this implies, respectively: $x = 1$ and $w = 3/5$). This value is fully compatible with neural networks analog implementation.

4.3. The neuron block, N

The neuron block applies to the normalised input differential balanced current $I^l_{\text{syn}_j}$ a sigmoid shape whose slope factor is programmable (i.e. it is set by the current $I_{\text{Slope Prog}}$). The output of the block N^l_j is the differential balanced current $I^l_{\text{neu}_j}$.

In this section, the slope-programmable translinear neuron circuit is presented [33]. The neuron is composed by two circuit blocks (see Fig. 8); a slope-programmable circuit and an activation circuit (able to apply a hyperbolic tangent shape to the output currents of the slope-programmable circuit) compose the neuron. It is possible to vary the sigmoid shape of the programmable neuron circuit by setting the external current $I_{\text{Slope Prog}}$. The slope-programmable translinear neuron block *N* is based on TL loops, which are evidenced in Fig. 8. Both circuits are based on the translinear principle.

4.3.1. The slope-programmable circuit

The slope-programmable circuit output current is given by: $I_{\text{Out}_{\text{Slope Prog}}} = I_2 - I_1 = xw2kI_B$.

The external current $I_{\text{Slope Prog}}$ controls the *k* parameter as follows: $k = I_{\text{Slope Prog}}/I_B$.

4.3.2. The activation circuit

The activation circuit applies a hyperbolic tangent shape to the current $I_{\text{Out}_{\text{Slope Prog}}}$. In Fig. 8, we can identify three translinear loops (which are evidenced by the square boxes).

The output current of the slope-programmable translinear neuron (block *N*), $I_{\text{neu}} = I^+_{\text{neu}} - I^-_{\text{neu}}$, is given by:

$$I_{\text{neu}} = I^+_{\text{neu}} - I^-_{\text{neu}} = I_B \frac{xwk}{(xwk)^2 + 2}\sqrt{(xwk)^2 + 4}$$

The previous function exhibits a hyperbolic tangent shape as shown in Fig. 9.

In Fig. 10, the simulated output current I_{neu} of the neuron circuit as function of the *x* input with the pa-

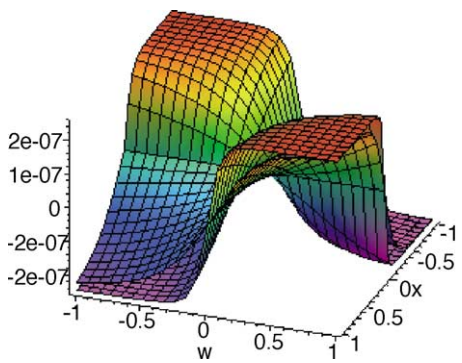

Fig. 9. The hyperbolic tangent shape of the slope-programmable translinear neuron (block *N*) drawn for $k = 5$ (sigmoid shape) and $k = 20$ (hard limiter shape).

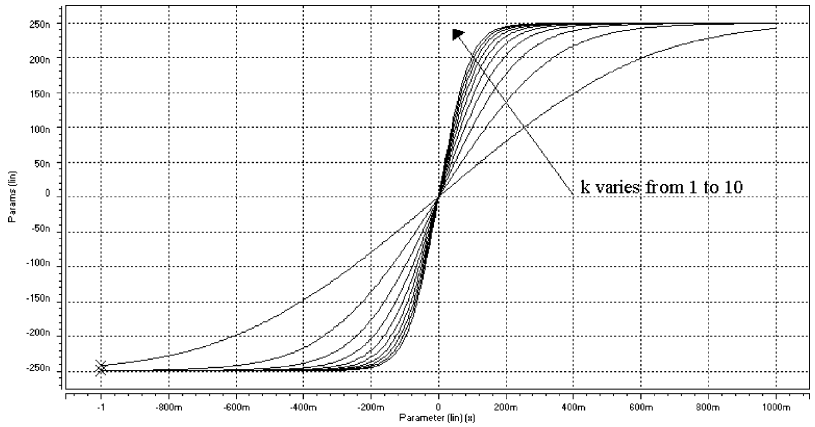

Fig. 10. Simulated output current I_{neu} varying x from -1 to 1 and $w = 1$, and varying k from 1 to 10.

rameter w set at value $+1$, varying the controlling-slope k parameter from 1 to 10, is shown. The current I_{neu} varies in the range $[-250\,\text{nA}, +250\,\text{nA}]$.

4.4. Ancillary circuit blocks

4.4.1. The target transconductor block, GT

The target transconductor block codifies the single-ended voltage targets $V^2_{\bar{X}_j}$ into a differential balanced current $I^2_{T_j}$ (please note that the superscript 2 refers to the second (output) neuron layer). The circuit schema is equal to the circuit schema of the weight transconductor G_W. The output current is: $I_{\text{target}} = I^+_{\text{T}} - I^-_{\text{T}} = (1+t)I_{\text{B}}/2 - (1-t)I_{\text{B}}/2 = tI_{\text{B}}$ where $t = 2g_{\text{m}}V^2_{\bar{X}}/I_{\text{B}}$ is the target value. Please, note that $-1 \le t \le 1$ and that g_{m} is the transconductance of the transistors of the input differential pair.

4.4.2. The block D

I_{neu} and I_{T} are differential and balanced currents:

$$I_{\text{neu}} = I^+_{\text{neu}} - I^-_{\text{neu}} = \tfrac{1}{2}(1+y)I_{\text{B}} - \tfrac{1}{2}(1-y)I_{\text{B}}$$

$$I_{\text{T}} = I^+_{\text{T}} - I^-_{\text{T}} = \tfrac{1}{2}(1+t)I_{\text{B}} - \tfrac{1}{2}(1-t)I_{\text{B}}$$

where y (neuron output value) and t (target value) are parameters that can vary in the range $[-1, +1]$ and $I_{\text{B}} = 250\,\text{nA}$.

The block D computes the absolute value of the difference between and $I^2_{\text{T}_j}$ and $I^2_{\text{neu}_j}$. It gives as output $I_{|\Delta\varepsilon_{pj}|} = |I^2_{\text{T}_j} - I^2_{\text{neu}_j}|$.

4.4.3. The error gradient measure block, EGM

The error gradient measure block (EGM) provides each synapse with V_{sign} (as a voltage signal) and the current coding the absolute value of the error function gradient (i.e. $I_{|\Delta\varepsilon_p|}$) (see points 7 and 8 of the algorithm of Table 1).

The inputs of the EGM block are:

- the current value coding the error gradient *before* the weights perturbation (i.e. I_{ε_p}(nopert)) and,
- the current value coding the error gradient after the weights perturbation (i.e. I_{ε_p}(pert)).

Both these currents are stored in two different current memory cells (blocks MC).

The error gradient measure block EGM, computes and backward propagates to the synapses S^l_{ji} the signals V_{sign}, and $I|\Delta\varepsilon|$inc and $I|\Delta\varepsilon|$dec. The details of the circuit implementation are given in [35].

4.4.4. The current memory cell, block MC

The current memory cell is used to store the current that codifies the value of the error function before and after the weight perturbation, i.e. the current $I_{|\Delta\varepsilon_p|}$. The adopted technique for the circuit implementation refer to "switched currents" [36], current copier cells" [37], and "dynamic current mirrors" [38].

The time response of the current memory cell is about 400 ns. $I_{|\Delta\varepsilon_p|}$ is in input to the current memory cell. The output current I_{stored} is in the range $I_{|\Delta\varepsilon_p|}$ $\pm 5\,\text{nA}$.

5. Results

We extensively simulated (using HSPICE [39]) at transistor level the netlist extracted from the physical design the two-layer MLP network, supplied by a voltage of ± 1.25 V.

We report here the results of the learning of the XOR function even if it is a well-known toy problem. Nevertheless, the learning of the logic EXOR function is an extremely challenging task for a neural network: in fact, the solution is not linearly separable. This test can prove the functioning of the network. In fact, parasitic capacitance/resistance and interference between neighbouring interconnections may crucially affect the circuit performance in analog VLSI. The circuit simulations of the learning of the EXOR function took many CPU hours (i.e. days): the simulation at circuit level of more complex learning tasks is practically unfeasible; in particular if the network is the one extracted from the physical design (as is has been done for all simulations results reported in this paper). A $2 \times 3 \times 1$ MLP network has been used. The number of synapses is 9.

All the weight voltages have been initially set to small random values. The value of V_{step} and the time duration of *Enable_update* have been, respectively, set to 20 mV and 100 ns. The input and the target signals timing are shown in Fig. 11. After about 800 epochs at the average, the network learned the EXOR logic function. Please note that the learning task in this case is more difficult also due to the by-pattern operation (see Section 2).

In Fig. 12, the output current and the target current in the last iterations before convergence in sample learning simulation are reported.

Table 6 reports the simulation results: the results demonstrate the capability of the network to learn from training examples. In all the experiments, the network was able to learn the target logic function.

The physical design of the circuits and of the overall architecture has been fully developed. The silicon area of the synapse is of about 0.015 mm^2 and the one of the neuron cell is of about 0.01 mm^2. The power consumption of the synapse is of about 2 μW and the one of the neuron is about 2 μW. The previous figures demonstrate the area and power efficiency of the proposed circuit implementation.

Usually it is not viable to compare the performance of analog on-chip learning neural networks because of the lack of data in many papers. Moreover, it is very difficult to evaluate how the implementation technology (i.e. its minimum feature size) determines the performance. However, in general, analog VLSI designs do not exploit at best the small feature size provided by technology; in fact, using non-minimum dimensions, the designer can reduce random as well as systematic errors.

To obtain a meaningful comparison, we adopt the metrics introduced in [40]. The computational power (connection updates per second (CUPS)) is

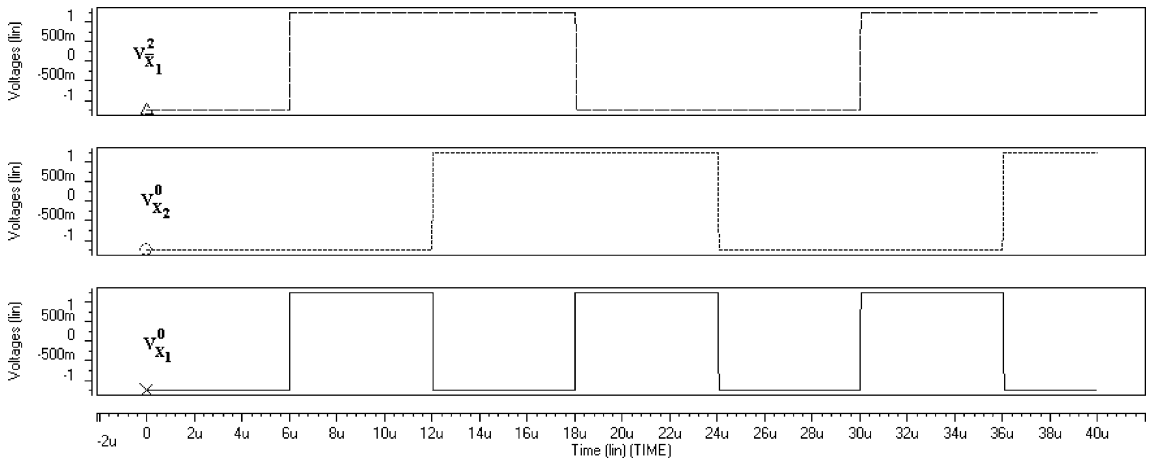

Fig. 11. Input ($V^0_{X_1}$, $V^0_{X_2}$) and target ($V^2_{X_1}$) signals timing for the learning of the EXOR logic function.

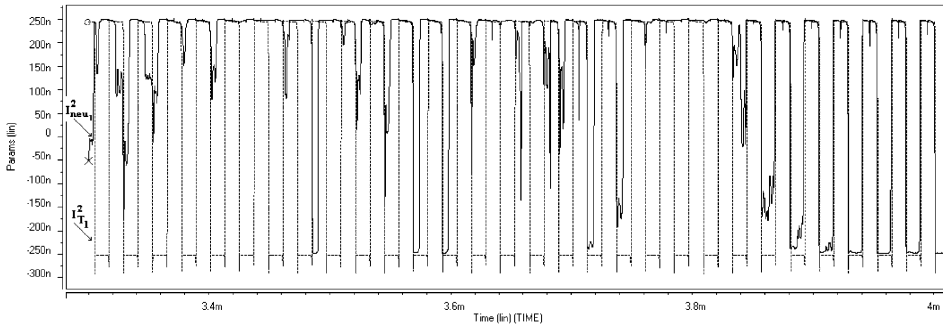

Fig. 12. $I^2_{\text{neu}_1}$ and $I^2_{\text{T}_1}$ waveforms in the last iterations before convergence (sample simulation).

Table 6
Learning simulation results for the XOR problem

Network topology	Iteration time (μs)	V_{step} (mV)	Time duration of *Enable_update* (ns)	Convergence (average figures)	
				Number of epochs	Time (ms)
2 × 3 × 1	6	20	100	800	19.2

normalised with respect to the power consumption (power efficiency: CUPS/mW) and to the silicon area (computational density: CUPS/mm^2). These figures also provide a realistic indication of the scalability of the architecture with respect to the silicon area and the power consumption. Moreover, to compare the performance with digital signal processors, the number of Operations (OP i.e. multiplications, sums, subtractions, power, etc.) were evaluated per connection update (i.e. OP per CU), the power efficiency was translated in terms of mW per OPS (i.e. operations per second, mW/OPS) and the computational density in terms of mm^2 per OPS (i.e. mm^2/OPS). In fact, it is well known that the number of instructions per second (i.e. IPS) is a usual parameter for evaluating DSP performance [41]. Analog on-chip learning neural processors implement only arithmetic operations. The counterpart for IPS is the OPS.

In the following, we compare our results with those discussed in [1] where a thorough analysis of the performance of analog VLSI neural chips with on-chip supervised learning has been reported. In doing so, we consider 2 × 3 × 1 MLP network topology for the analog architecture proposed in this paper.

Table 7
Comparison between analog on-chip supervised learning networks chips (and DSPs)

Chip	Technology	Computational power		
		CUPS	OP per LI	OPS
[42] (two chips)	Standard CMOS 2.4 μm	0.27×10^6	140	37.8×10^6
[43]	CMOS 1 μm	4.6	49	225
[17]	CMOS, double poly, double metal, 2 μm	42×10^3	367	15.414×10^6
[44,45]	CMOS double poly, double metal, 2 μm	80×10^3	21	1.680×10^6
[46]	CMOS double poly, double metal, 1.3 μm	10×10^6	522	5220×10^6
[47,48]	CMOS double metal, single poly, 0.7 μm	2.65×10^6	704	1865.6×10^6
[49]	CMOS double metal, double poly, 1.2 μm	7.5×10^6	69	517.5×10^6
This paper	CMOS double metal, double poly, 0.8 μm	1.5×10^6	56	1.5×10^6
[41]	CMOS 0.18 μm	Not applicable	Not applicable	5×10^3 (MIPS)
[50] TMS320C5000 platform	Not available	Not applicable	Not applicable	600 (MIPS)

Table 8
Comparison between analog on-chip supervised learning networks chips (and DSPs)

Chip	Computational density		Energy efficiency	
	CUPS/mm^2	mm^2/OPS	CUPS/mW	mW/OPS
[42] (two chips)	22.5×10^3	317×10^{-9}	13.5×10^3	529×10^{-9}
[43]	1.15	178×10^{-3}	92	222×10^{-6}
[17]	8.7×10^3	313×10^{-9}	35×10^3	79×10^{-9}
[44,45]	2.7×10^6	18×10^{-9}	133×10^3	358×10^{-9}
[46]	204×10^3	9×10^{-9}	32×10^3	60×10^{-9}
[47,48]	216×10^3	7×10^{-9}	106×10^3	13×10^{-9}
[49]	870×10^3	17×10^{-9}	37.5×10^3	386×10^{-9}
This paper	5×10^6	33×10^{-9}	30×10^6	6×10^{-9}
[41]	Not applicable	0.5 (mm^2/MIPS)	Not applicable	0.1 (mW/MIPS)
[50] TMS320C5000 platform	Not applicable	Not available	Not applicable	0.05 (mW/MIPS)

The performance in terms of computational power[2], are compared in Table 7.

The performance in terms of computational density and power efficiency, are compared in Table 8.

As it can be seen from previous comparisons, the proposed architecture is very efficient in terms of computational density and, what is more, in term of power efficiency. Moreover, with respect to the implementations cited in Tables 7 and 8, it exhibits better capabilities in terms of modularity and scalability.

6. Conclusion

In this paper, we have addressed the analog VLSI implementation of supervised on-chip learning networks with the goal of efficiency in terms of scalability, modularity, computational density and power consumption.

Scalability and modularity were addressed both at the system as well as at the circuit level. In the former case, it was accomplished by adopting WP algorithms, which are more prone to scalable architectures. In the latter case, current-mode circuits were adopted. Moreover, the current mode approach increases the dynamic range of signals and, with weak inversion biased circuits, lowers power consumption.

We have presented in this paper: (i) novel circuits for the implementation of neural primitives; (ii) a novel analog VLSI circuit architecture implementing on-chip a weight perturbation learning algorithm. To increase modularity and linearity, a differential and balanced current mode signalling approach was adopted. The neuron and synapse cells were implemented using translinear circuits. The whole circuit architecture was designed up to the physical level using a CMOS 0.8 μm technology.

We simulated intensively the architecture at transistor level (e.g. HSPICE) using the netlist extracted from the physical design.

The performance of the analog circuit architecture compare favourably with those of the open literature. In particular, the proposed architecture is very efficient in terms of computational density and, what is more, in term of power efficiency. Moreover, it exhibits better capabilities in terms of modularity and scalability.

We have not addressed the implementation of the long term storage on-chip. This because this issue can be solved by the use of non-volatile memories (NVM) [1]: given the learning feedback scheme, read mode circuits are not necessary and the control of the stored charge can be straightforwardly implemented by exploiting the learning feedback. NVM has been successfully used in many proposed implementations and in computing arrays. The NVM is easily interfaced with learning circuits and the programming phase (which is time consuming) can be carried out only at the end of the learning or interleaved with learning iterations.

Our results demonstrate that by adopting perturbation based learning algorithms and dedicated circuit design approaches, very efficient analog (and eventually mixed-mode) systems can be achieved with the goal of the VLSI implementation of cognitive systems

[2] LI, learning iteration.

(e.g. learning algorithms) and smart adaptive systems on silicon.

References

[1] M. Valle, Analog VLSI Implementation of Artificial Neural Networks with Supervised On-chip Learning, to appear on Analog Integrated Circuits and Signal Processing, Kluwer Academic Publishers, The Netherlands, December 2002.

[2] D. Anguita, M. Valle, Perspectives on dedicated hardware implementations, in: Proceedings of the European Symposium on Artificial Neural Networks, Bruges, Belgium, 25–27 April 2001, pp. 45–55.

[3] A.H. Kramer, Array-based analog computation, IEEE Micro 16 (5) (1996) 20–29.

[4] R. Genov, G. Cauwenberghs, Charge-mode parallel architecture for vector-matrix multiplication, IEEE Trans. CAS II—Analog Digital Signal Process. 48 (10) (2001) 930–936.

[5] http://www.austriamicrosystems.com.

[6] S. Haykin, Neural Networks: a Comprehensive Foundation, second ed., Prentice Hall, 1999.

[7] L. Tarassenko, A Guide to Neural Computing Applications, Arnold Publishers, London, 1999.

[8] J. Hertz, A. Krogh, R.G. Palmer, Introduction to the Theory of the Neural Computation, Addison-Wesley Publishing Company, 1991.

[9] A. Cichocki, R. Unbehauen, Neural Networks for Optimisation and Signal Processing, Wiley, 1993.

[10] M. Valle, D.D. Caviglia, G.M. Bisio, An Analog VLSI Neural Network with On-Chip Back Propagation Learning, Analog Integrated Circuits and Signal Processing, Kluwer Academic Publisher, vol. 9, 1996, pp. 231–245.

[11] M. Jabri, B. Flower, An optimal architecture and learning technique for analog VLSI feedforward and recurrent multilayer networks, IEEE Trans. Neural Networks 3 (1) (1992) 154–157.

[12] J. Alspector, D. Lippe, A study of parallel perturbative gradient descent, Adv. Neural Inf. Process. Syst. 7 (NIPS'94) (1995) 803–810.

[13] M.A. Jabri, R.F. Coggins, B.G. Flower, Adaptive Analog VLSI Neural Systems, Chapman & Hall, 1996.

[14] G. Cauwenberghs, A fast stochastic error-descent algorithm for supervised learning and optimization, Adv. Neural Inf. Process. Syst. 5 (NIPS5) (1993) 244–251.

[15] J. Alspector, R. Meir, B. Yuhas, A. Jayakumar, D. Lippe, A parallel gradient descent method for learning in analog VLSI neural networks, Adv. Neural Inf. Process. Syst. 5 (NIPS5) (1993) 836–844.

[16] A. Annema, Feed-Forward Neural Networks: Vector Decomposition Analysis, Modelling and Analog Implementation, Kluwer Academic Publishers, 1995.

[17] G. Cauwenberghs, An analog VLSI recurrent neural network learning a continuous-time trajectory, IEEE Trans. Neural Networks 7 (2) (1996) 346–361.

[18] G. Cairns, L. Tarassenko, Precision issues for learning with analog VLSI multilayer perceptrons, IEEE Micro 15 (3) (1995) 54–56.

[19] M. Giudici, F. Queirolo, M. Valle, Evaluation of gradient descent learning algorithms with adaptive and local learning rate for recognising hand-written numerals, in: Proceedings of the 10th European Symposium on Artificial Neural Networks, ESANN'02, Bruges, Belgium, 24–26 April 2002, pp. 289–294 (ISBN 2-930307-02-1).

[20] M. Giudici, F. Queirolo, M. Valle, Stochastic supervised learning algorithms with local and adaptive learning rate for recognising hand-written characters, Presented at the International Conference on Artificial Neural Networks, ICANN 2002, Madrid, Spain, 27–30 August 2002. Springer-Verlag, Berlin Heidelberg, pp. 619–624 (ISBN 3-540-44074-7).

[21] F. Diotalevi, G.M. Bo, D.D. Caviglia, M. Valle, Evaluation and validation of local and adaptive weight perturbation learning algorithms for optical character recognition applications, in: Proceedings of the Third International ICSC Symposia on Intelligent Industrial Automation (IIA'99) and Soft Computing (SOCO'99), ICSC Academic Press, Canada/Switzerland, Genova, 1–4 June 1999, pp. 508–512.

[22] F. Diotalevi, M. Valle, D.D. Caviglia, Evaluation of gradient descent learning algorithms with an adaptive local rate technique for hierarchical feed-forward architectures, in: IEEE-INNS-ENNS International Joint Conference on Neural Networks, IEEE Press, Piscataway, NJ, Como, Italy, 24–27 July 2000.

[23] F. Diotalevi, M. Valle, Weight perturbation learning algorithm with local learning rate adaptation for the classification of remote-sensing images, in: Proceedings of the 9th European Symposium on Artificial Neural Networks, ESANN'01, Bruges, Belgium, 25–27 April 2001, pp. 217–222.

[24] A. Nève, D. Flandre, J.J. Quisquater, Feasibility of smart cards in silicon-on-insulator (SOI) technology, in: USENIX Workshop on Smartcard Technology (Smartcard'99), 1999, pp. 1–7.

[25] F. Diotalevi, M. Valle, G.M. Bo, E. Biglieri, D.D. Caviglia, An analog on-chip learning circuit architecture of the weight perturbation algorithm, in: ISCAS'2000, IEEE International Symposium on Circuits and Systems, IEEE Press, Piscataway, NJ, Geneva, Switzerland, 28–31 May 2000, pp. II 717–II 720.

[26] C. Toumazou, J.B. Hughes, D.M. Pattullo, Regulated cascode switched-current memory cell, Electron. Lett. 25 (5) (1990) 303–305.

[27] B. Gilbert, Translinear circuits: a proposed classification, Electron. Lett. 11 (1) (1975) 14–16;
B. Gilbert, Translinear circuits: a proposed classification, Electron. Lett. 11 (6) (1975) 136.

[28] E. Seevinck, Analysis and Synthesis of Translinear Integrated Circuits, Studies in Electrical and Electronic Engineering, vol. 31, Elsevier, Amsterdam, 1988.

[29] A.G. Andreou, in: B. Sheu, E. Sanchez-Sinencio, M. Ismail (Eds.), Low Power Analog VLSI Systems for Sensory Information Processing, Microsystems Technologies for Multimedia Applications: an Introduction, IEEE Press, Los Alamitos, CA, 1995.

[30] A.G. Andreou, K.A. Boahen, Translinear circuits in subthreshold MOS, J. Analog Integr. Circuits Signal Process. 9 (1996) 141–166.

[31] K.A. Boahen, Retinomorphic vision systems, in: Proceedings of the MicroNeuro-96, IEEE Computer Society Press, Los Alamitos, CA, 1996.
[32] F. Krummenacher, N. Joehl, A 4-MHz CMOS continuous-time filter with on-chip automatic tuning, IEEE J. Solid-State Circuits 23 (3) (1988) 750–758.
[33] F. Diotalevi, M. Valle, An analog CMOS four quadrant current-mode multiplier for low power artificial neural networks implementation, in: 15th European Conference on Circuit Theory and Design, ECCTD'01: "Circuit Paradigm in the 21st Century", Helsinki University of Technology, Espoo, Finland, 28–31 August 2001, pp. III 325–III 328.
[34] F. Diotalevi, M. Valle, G.M. Bo, E. Biglieri, D.D. Caviglia, Analog CMOS current mode neural primitives, in: ISCAS'2000, IEEE International Symposium on Circuits and Systems, IEEE Press, Piscataway, NJ, Geneva, Switzerland, 28–31 May 2000, pp. I 419–I 422.
[35] F. Diotalevi, Analog Microelectronic Supervised Learning Systems, Ph.D. thesis, 2001, http://www.micro.dibe.unige.it/works/FDiotalevi_PHD_Thesis.zip.
[36] J.B. Hughes, N.C. Bird, I.C. MacBeth, Switched-currents, a new technique for analogue sampled-data signal processing, in: IEEE International Symposiums on Circuits and Systems, 1989, pp. 1584–1587.
[37] D. Vallencourt, Y.P. Tsividis, S. Daubert, Sampled-current circuits, IEEE International Symposiums on Circuits and Systems, 1989, pp. 1592–1595.
[38] G. Wegman, E.A. Vittoz, Very accurate dynamic current mirrors, Electron. Lett. 25 (10) (1989) 644–646.
[39] HSPICE, 2003, http://www.synopsys.com/products/mixedsignal/hspice_ds.html.
[40] A.H. Kramer, Array-based computation: principles, advantages and limitations, in: Proceedings of the Microneuro'96, 1996, pp. 68–79.
[41] G. Frantz, Digital signal processor trends, IEEE Micro 20 (6) (2000) 52–59.
[42] T. Lehmann, Hardware learning in analog VLSI neural networks, Ph.D. thesis, Electronics Institute, Technical University of Denmark, 1994 (http://eivind.imm.dtu.dk/publications/phdthesis.html).
[43] Y. Berg, R.L. Sigvartsen, T.S. Lande, A. Abusland, An analog feed-forward neural network with on-chip learning, Analog Integr. Circuits Signal Process. 9 (1996) 65–75.
[44] A.J. Montalvo, et al., An analog VLSI neural network with on-chip perturbation learning, IEEE J. Solid State Circuits 32 (4) (1997) 535–543.
[45] A.J. Montalvo, et al., Towards a general-purpose analog VLSI Neural Network with on-chip learning, IEEE Trans. Neural Networks 8 (2) (1997) 413–423.
[46] T. Morie, Analog VLSI implementation of self learning neural networks, in: G. Cauwenberghs, M. Bayoumi (Eds.), Learning on Silicon-Adaptive VLSI Neural Systems, Kluwer Academic Publishers, 1999.
[47] G.M. Bo, H. Chiblè, D.D. Caviglia, M. Valle, Analog VLSI on-chip learning neural network with learning rate adaptation, in: G. Cauwenberghs, M.A. Bayoumi (Eds.), Learning on Silicon-Adaptive VLSI Neural Systems, The Kluwer International Series in Engineering and Computer Science, vol. 512, Kluwer Academic Publishers, June 1999, pp. 305–330.
[48] G.M. Bo, D.D. Caviglia, M. Valle, A self-learning analog neural processor, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Tokyo, Japan, vol. E85-A, no. 9, pp. 2149–2158, September 2002 (ISSN 0916-8508).
[49] C. Lu, B. Shi, L. Chen, An on-chip BL learning neural network with ideal neuron characteristics and learning rate adaptation, Analog Integr. Circuits Signal Process. 31 (2002) 55–62.
[50] TMS, 2001, http://dspvillage.ti.com/docs/dspproducthome.jhtml.