Noname manuscript No.
(will be inserted by the editor)

Efficient parallel implementation of reservoir computing systems

M. L. Alomar · Erik S. Skibinsky-Gitlin · Christiam F. Frasser · Vincent Canals · Eugeni Isern · Miquel Roca · Josep L. Rosselló

Received: date / Accepted: date

Abstract Reservoir computing (RC) is a powerful machine-learning methodology well suited for time-series processing. The hardware implementation of RC systems (HRC) may extend the utility of this neural approach to solve real-life problems for which software solutions are not satisfactory. Nevertheless, the implementation of massive parallel-connected reservoir networks is costly in terms of circuit area and power, mainly due to the requirement of implementing synapse multipliers that increase gate count to prohibitive values. Most HRC systems present in the literature solve this area problem by sequencializing the processes, thus loosing the expected fault-tolerance and low latency of fully parallel-connected HRCs. Therefore, the development of new methodologies to implement fully-parallel HRC systems is of high interest to many computational intelligence application requiring quick responses. In this article, we propose a compact hardware implementation for Echo-State Networks (an specific type of reservoir) that reduces the area cost by simplifying the synapses and using linear piece-wise activation functions for neurons. The proposed design is synthesized in a Field-Programmable Gate Array and evaluated for different time-series prediction tasks. Without compromising the overall accuracy, the proposed approach achieves a significant saving in terms of power and hardware when compared with recently published implementations. These techniques pave the way for the low power implementation of fully-parallel reservoir networks containing thousands of neurons in a single integrated circuit.

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness (MINECO), the Regional European Development Funds (FEDER), and the Comunitat Autnoma de les Illes Balears under grant contracts TEC2014-56244-R, TEC2017-84877-R and a fellowship (FPI/1513/2012) financed by the European Social Fund (ESF) and the Govern de les Illes Balears (Conselleria d'Educació, Cultura i Universitats).

The authors are with the Electronics Engineering Group at Department of Physics, University of Balearic Islands, Ctra. Valldemossa Km 7.5, Palma de Mallorca 07122, Spain
E-mail: Josep L. Rosselló
j.rossello@uib.com

Keywords artificial neural networks · recurrent neural networks · reservoir computing · echo sate networks · hardware neural network · field-programmable gate array · time-series prediction

1 Introduction

Although the majority of artificial neural networks (ANNs) are implemented in software using conventional processors, numerous applications require the use of specific hardware [1]. Unlike general-purpose sequential processors, specific integrated circuits realizing ANN models can take advantage of the inherent parallelism in the neural processing. In general, specialized ANN hardware supporting or replacing software may be beneficial in terms of speed, power requirements, reliability and cost [2, 3]. For example, hardware neural networks (HNNs) can be used to speed up the neural processing in high-volume real-time applications, such as computer vision tasks [4], image search [5] and data mining [6]. On the other hand, energy-efficient HNNs are demanded in autonomous/mobile applications constrained in terms of power supply, such as the control of machines and industrial processes [7], distributed sensory networks [8], portable medical applications [9] or handwriting and speech recognition systems [10]. A great research effort has been made to develop efficient HNN implementations, which has been mainly focused on the design of the non-linear sigmoidal activation function using either digital [11, 12, 13] or analogue [14] circuits. However, the application of the internal weights present in ANNs sharply constrains the parallel implementation of massive networks in a single chip. This is evident when using digital hardware, which requires large resources, and especially in field-programmable gate arrays (FPGAs) [15]. The use of approximate multipliers [16] and stochastic computing [15] has been proposed as possible solutions to reduce the hardware requirements at the cost of accuracy loss. Nevertheless, this shortcoming is assumable when dealing with computational intelligence applications in which input data may present large degrees of uncertainty and the processing to be done consists in recognizing generic patterns.

Reservoir computing (RC) is an affordable technique to implement and train recurrent neural networks (RNNs) that is suited to processing temporal information and providing outstanding performance on time-series prediction and classification tasks [17]. RC has been successfully applied in numerous domains, such as robot control [18], image/video processing [19], wireless sensor networks [20], financial forecasting [21] or the monitoring of physiological signals [22]. Fast and efficient hardware designs implementing RC systems can be interesting for these applications, which require real-time intensive data processing and/or the use of low-power devices to ensure a long battery lifetime. Numerous implementations of the RC concept have been proposed through the use of unconventional hardware platforms [17] and even some FPGA realizations [23, 24]. As a drawback, most of them are sequentially operated to enable a reduced use of resources, which compromises the processing speed.

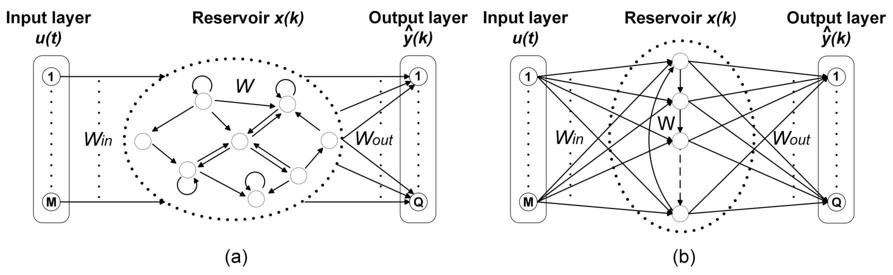

Fig. 1 (a-b) Reservoir Computing architectures: (a) general architecture of RC where all connections ($\boldsymbol{W_{in}}$ and $\boldsymbol{W}$) are random and fixed except those from the reservoir to the output units ($\boldsymbol{W_{out}}$), which are trained for each specific task; (b) reservoir with simple cyclic topology

The reservoir computing scheme belongs to a wider category, the ANN with non-iterative learning that present a much more simple and fast configuration methodology if compared with other types of networks.

In this article, we present a fully parallel implementation realizing large reservoir computing systems with very few hardware resources. The proposed digital hardware design presents a low power dissipation and a reduced number of hardware resources when compared to conventional digital solutions. As a demonstration of the validity of the approach, we implement a large reservoir network within an FPGA and evaluate its performance for three traditional benchmarks on time-series processing (Santa-Fe and Mackey Glass chaotic oscillator prediction tasks and the equalization of a non-linear communication channel). The results are compared with previously-published hardware implementations of Reservoir Computing systems. We summarize the contributions of this work as follows:

- We propose a compact methodology for the hardware implementation of Echo-State Networks (ESN).
- The proposed model enable the implementation of fully parallel-connected ESN, thus implying a higher processing speed if compared with sequentially-based solutions.
- The proposed methodology present an accuracy similar than other hardware applications while overall performance (speed-power) is significantly enhanced.

2 Related work

Different approaches have been proposed for the implementation of Reservoir computing in electronic systems, most of them based on sequential architectures architectures. Three main types of implementing technologies can be differentiated (analog, digital and optoelectronic). In the work implemented by Soriano et al. [25] we find a mixed analog/digital solution using a delayed-feedback architecture. The incoming signal is pre-processed using a mask and converted to an analog signal to stimulate a Mackey-Glass analog oscillator that use two operational amplifiers. The output signal of the oscillator is digitalized and stored in a register to be feedback with the incoming signal. The output is also digitally post-processed to evaluate the result of the reservoir. The model is applied with success to speech recognition and for time-series forecasting. The main drawback of this approximation is the excessive delay present within the analog block if compared with fully digital implementations as the the work proposed by Alomar et al. [24]. In this study, a fully digital version of the Soriano's design is implemented with the difference of substituting the analog non-linear oscillator by a digital one. The proposed model is applied to time-series forecasting and temporal series recognition. The work present a low power and a moderate speed due to the sequential nature of the architecture (based on the implementation of the delay-based oscillator). A fully-parallel FPGA implementation is developed later by the same authors [23] in which an stochastic approach is used. Stochastic computing is an emerging unconventional processing methodology that substitute the wide bus connectors used by traditional binary circuits by single switching bits (coding the stochastic signals). With this technique, complex binary multipliers can be substituted by single logic gates and therefore a fully-connected network with thousands of interconnected neurons can be easily built. The main drawback of this approach is the long latency period to convert the switching signals to binary ones (needed when a given result must be properly stored). Other works are based on an optoelectronic implementation [26, 27, 28], also dependent on a sequential architecture and normally applied to time-series forecasting and speech recognition. Those designs present an impressive processing speed due to their photonic nature, nevertheless the full experimental setting may present a considerable power dissipation. Recently, a fully-parallel photonic recurrent neural network has been implemented using an optoelectronic technology for ultra-fast signal processing [29]. In this work, up to 2025 diffractively coupled photonic nodes is implemented in a large-scale recurrent neural network. The model provide a good performance in terms of computational efficiency.

Therefore, most research in RC circuit implementation is based on the sequentializing the process, thus limiting the processing speed. All the above mentioned implementations are compared in the tables with the proposed model in terms of accuracy, power and processing speed (when this information is available in the references).

3 Methodology

3.1 Notation

We first define some common notation[30, 31, 32] followed in this work. In particular, we use bold capital letters (e.g. $\mathbf{X}$) and bold lowercase letters (e.g. $\mathbf{x}$) to denote matrices and vectors respectively. We employ non-bold letters (e.g. x) to represent scalars, and Greek letters (e.g. β) to represent parameters.

All vectors are assumed to be in column form. Finally, the hat symbol is used to denote approximations performed by the machine learning system (e.g. $\hat{\mathbf{y}}$ is an approximation of $\mathbf{y}$).

3.2 Echo-State Networks principles

ESN differs from previous RNN techniques in that conventional RNNs tune all the synaptic weights between neurons to perform specific tasks whereas in ESN most weights are randomly chosen and kept fixed (see Fig.1a). Only the connections from the fixed RNN (termed the reservoir) to a non-recurrent output layer (the readout) are modified by learning. This strategic design avoids the need for complex back-propagation of the errors over the dynamic part of the network reducing the training to a classical linear regression problem.

The architecture of a reservoir computing system consists of a total of N internal processing nodes (the neurons) each one providing a given value $x_i(k)$, where $i \in \{1, 2, .., N\}$ is the neuron index, k represents the evolution during time ($k \in \{1, 2, ..., L\}$) and L is the total number of samples taken from the reservoir. Therefore, the time evolution of internal nodes of the reservoir is described by a matrix with L rows and N columns $\boldsymbol{X}$ (the design matrix). The state of the network at a given time k is defined by the kth row of the design matrix and the time evolution of the ith neural output is stored in the ith column. The output response of the reservoir is computed in two phases: First, the current reservoir state $[\boldsymbol{x}(k)]$ is updated according to a nonlinear function of the weighted sum of the neuron inputs [M external time-dependent inputs $\boldsymbol{u}(k) = (u_1(k), u_2(k), ..., u_M(k))^T$], and N internal ones coming from the reservoir's neurons evaluated in the previous time step, following the next expression:

$$\boldsymbol{x}(k) = f\left[\boldsymbol{W_{in}}\boldsymbol{u}(k) + \boldsymbol{W}\boldsymbol{x}(k-1)\right] \tag{1}$$

where f is a non-linear function (called the activation function $f : \mathbb{R}^N \rightarrow \mathbb{R}^N$) while $\boldsymbol{W_{in}}$ and $\boldsymbol{W}$ are two $N \times M$ and $N \times N$ weight matrices respectively. In a second phase, a total of Q outputs $[\hat{\boldsymbol{y}}(k) = (\hat{y}_1(k), \hat{y}_2(k), ..., \hat{y}_Q(k))^T]$ are obtained as a linear combination of the reservoir states:

$$\hat{\boldsymbol{y}}(k) = \boldsymbol{W_{out}}\boldsymbol{x}(k) \tag{2}$$

where $\boldsymbol{W_{out}}$ is a $Q \times N$ weight matrix that is obtained using a linear regression with respect the expected outputs $[\boldsymbol{y}(k) = (y_1(k), y_2(k), ..., y_Q(k))^T]$.

3.3 Training methodologies

Defining $\boldsymbol{Y}$ as the feature matrix of $L \times Q$ to be approximated by the network (composed of L row vectors of Q elements $\boldsymbol{y}(k)$), we have that $\boldsymbol{W_{out}}$ may be estimated using the Moore-Penrose pseudo-inverse:

$$\boldsymbol{W_{out}^T} = \left(\boldsymbol{X^T}\boldsymbol{X}\right)^{-1}\boldsymbol{X^T}\boldsymbol{Y} \tag{3}$$

In the reservoir computing scheme, matrices $\boldsymbol{W_{in}}$ and $\boldsymbol{W}$ are taken fixed while $\boldsymbol{W_{out}}$ is conveniently trained using expression 3 or other similar linear fitting. If some columns of the design matrix $\boldsymbol{X}$ are linearly dependent, the maximum possible number of elements of the weight matrix are set to zero to obtain a basic solution, and only the independent columns are considered. To remove the dependent columns of matrix $\boldsymbol{X}$, the rank-revealing QR decomposition is used. Therefore, the design matrix is factorized $\boldsymbol{X} = \boldsymbol{QR}$, where $\boldsymbol{Q}$ is the orthogonal matrix and $\boldsymbol{R}$ an upper triangular matrix. The weight coefficients associated to the $\boldsymbol{X}$ dependent columns are equated to zero. For this special case, the nonzero widths are calculated as:

$$\boldsymbol{W_{out}^{T}}\left(\boldsymbol{\xi}\right) = \hat{\boldsymbol{R}}^{-1}\hat{\boldsymbol{Q}}^{\boldsymbol{T}}\boldsymbol{Y} \tag{4}$$

where $\hat{\boldsymbol{R}}$ and $\hat{\boldsymbol{Q}}$ are reduced versions of $\boldsymbol{R}$ and $\boldsymbol{Q}$ only considering the non-zero columns of $\boldsymbol{Q}$ and removing the rows and columns of $\boldsymbol{R}$ that are associated to the dependent columns of $\boldsymbol{X}$. Parameter ξ corresponds to a selection that only considers the independent components [33, 34]. Therefore, if r is the rank of matrix $\boldsymbol{X}$, then $\hat{\boldsymbol{R}}$ is a $r \times r$ square matrix and $\hat{\boldsymbol{Q}}$ is a $L \times r$ matrix. A new set of descriptors that are linearly independent can therefore be obtained as $\hat{\boldsymbol{X}} = \boldsymbol{X}(\xi) = \hat{\boldsymbol{Q}}\hat{\boldsymbol{R}}$.

More accurate linear regression models can be used as the Generalized Least Squares (GLS). Defining $\boldsymbol{\Omega}$ as the conditional variance of the error term given $\boldsymbol{X}$ ($\boldsymbol{\Omega} \equiv Cov[\epsilon|\boldsymbol{X}]$), a more accurate expression for $\boldsymbol{W_{out}}$ can be derived by minimizing the squared Mahalanobis length of the residual vector $\boldsymbol{e} \equiv \boldsymbol{Y} - \boldsymbol{W_{out}}\boldsymbol{X}$.

$$\boldsymbol{W_{out}} = \arg\min_{\mathbf{W}'} \left\{ (\boldsymbol{y} - \mathbf{W}'\boldsymbol{X})^{T} \boldsymbol{\Omega}^{-1} (\boldsymbol{y} - \mathbf{W}'\boldsymbol{X}) \right\} \tag{5}$$

The estimator therefore has the explicit expression of:

$$\boldsymbol{W_{out}^{T}} = (\boldsymbol{X}^{T}\boldsymbol{\Omega}^{-1}\boldsymbol{X})^{-1}\boldsymbol{X}^{T}\boldsymbol{\Omega}^{-1}\boldsymbol{Y} \tag{6}$$

This method is useful when having heteroscedasticity and autocorrelation in the error values $\boldsymbol{e}$. Theoretical and empirical discussion on differences between GLS and the Ordinary Least Square (OLS) methods can be found in [35].

3.4 Low cost hardware implementation of Echo-State Networks

The nonlinear activation function used in (1) is usually selected to be the hyperbolic tangent function. The number of neurons employed in ESNs is usually high (typically between 50 and 1000, although some applications require much larger networks to achieve the desired accuracy [19]), which makes particularly challenging the hardware implementation of these systems if parallelism has to be maintained. Given the large number of products to be implemented due to the high number of synapses, multipliers expend a significant portion of the integrated circuit resources if we want to use a parallel scheme. We propose to avoid the use of the synapses' multipliers by limiting the possible weights to integer powers of two and sums of powers of two so that shift registers and adders can be employed instead of multipliers. This technique has been widely used in other fields such as the digital filter design [36]. For an standard ANN implementation that use back-propagation as learning algorithm, the constraint on the weights is not desirable since it leads to lower network performance [37]. Nonetheless, we show that for reservoir networks with fixed connections, the proposed approach only implies a minor accuracy loss. Nevertheless, for the estimation of the output $[\hat{\boldsymbol{y}}(k)]$ from expression (2) we use the dedicated embedded multipliers integrated in the FPGA device. The overall area impact of this computation inside the FPGA is relatively low since the logic elements needed to implement the neural network are not used and the number of multipliers at the output layer is significantly lower than the synapses' multipliers (that would limit the maximum neural fan-in). Regarding the network topology, it has been shown that a deterministic reservoir with simple cyclic architecture (SCR, see Fig.1b) presents a similar performance compared to the classical random one. The SCR architecture minimizes the number of connections [38] and optimize the packing efficiency. In Fig.2 we show in detail the scheme of the SCR network implemented. For simplicity, the connections between internal units have the same weight r whereas the inputs are connected to the reservoir with a weight that is positive ($|\ v\ |$) or negative ($-\,|v|$) with the same probability and with the same absolute value. We define the vector $\epsilon = (\epsilon_1, \epsilon_2, \ldots, \epsilon_N)$ in which each component is randomly selected between two possible choices (-1 and $+1$) with equal probabilities. Then, the weight connection between the input ($u(t)$) and the i^{th} node of the reservoir is $\epsilon_i \cdot v$ (see Fig.2).

For the estimation of network configuration parameters r and v, we first adjust numerically to find the optimum weight configuration. Fig.3a illustrates a general circuit design for a two-input sigmoidal neuron necessary to build the cyclic reservoir of Fig.2. The fixed-point two's complement notation is assumed for all signals so that both positive and negative values can be represented. The first neuron's input $[u(t)]$ refers to the external input signal (to be processed by the network) and the second one $[x_{i-1}(t-1)]$ to the state of a neighboring neuron evaluated at the previous time step. A resolution of n bits is considered for the input and of m bits for the weights (v and r). The multiplier's output is truncated to n bits taking the most significant of the result, but a higher or lower resolution could be employed depending on the desired accuracy.

The general scheme of Fig.3a can be simplified to that of Fig.3b when the weights are limited to a few discrete values. In this case, the full multipliers are substituted by shift-and-add blocks. Such "multiplier-less" approach enable great hardware saving at the cost of constraining the possible values of the connection weights. The shift-and-add block is depicted in Fig.3c. Basically, it performs a multiplication of the input signal $[u(t)]$ by the corresponding weight (v) with a pair of shift registers and an adder. Some additional circuitry is included to perform the negation of the shifted values in case it is necessary. A multiplexer is employed to provide either the number that directly results

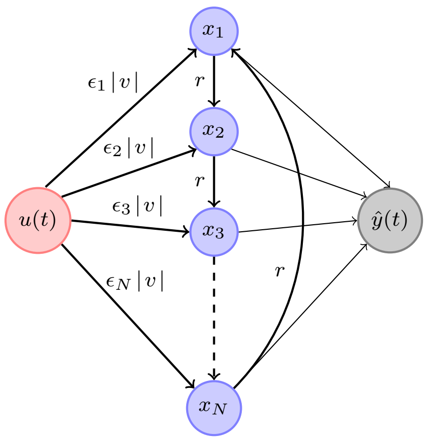

Fig. 2 The implemented cyclic Echo State Network scheme for the analysis of one-dimensional signals

from the shift register or its corresponding negative value depending on a selection signal. A decoder configures the shift registers (with the number of required shifts, $sh1$ and $sh2$) and controls the activation of the negations ($neg1$ and $neg2$) as a function of the configuration vector (ϵ). By way of example, a single right shift of the input ($sh1 = 1$) performs a multiplication by 0.5 while two shifts ($sh2 = 2$) are equal to a factor of 0.25. The direct addition (with $neg1 = neg2 = 0$, indicating that no negation of the shifted values is necessary) of these two shifted magnitudes results in a weight $v = 0.75$. The weight value $v = 0.875$ can be implemented by selecting no shifts and no negation for the first shift register ($sh1 = 0$, $neg1 = 0$) and three shifts with a negated output for the second one ($sh2 = 3$, $neg2 = 1$) so that the input signal $u(t)$ is weighted by the factor $v = 1 - 0.125 = 0.875$. A negative factor, for instance $v = -0.5$, may be obtained through the negation of both shifted magnitudes ($neg1 = neg2 = 1$), where each one is obtained with two displacements ($sh1 = sh2 = 2$) so that $v = -0.25+(-0.25) = -0.5$. Therefore, the possible weights are limited to:

$$\omega = neg_1 2^{-sh_1} + neg_2 2^{-sh_2} \tag{7}$$

In the case it is desired a generic design, the circuit of Fig.3c can be used. For FPGA implementations in which the training of r and v are done off-line and remain fixed in hardware, a fixed circuit design incorporating the implementation of the exact shifts for each neuron input can be used, further simplifying the hardware and increasing the processing speed.

A simple piece-wise linear approximation with three segments [11] is used for the implementation of the activation function, due to its simple binary implementation. More accurate designs (e.g., [12, 13]) could be employed to improve the network's performance at the cost of higher hardware requirements. It usually has a sigmoidal shape, however it may also take the form

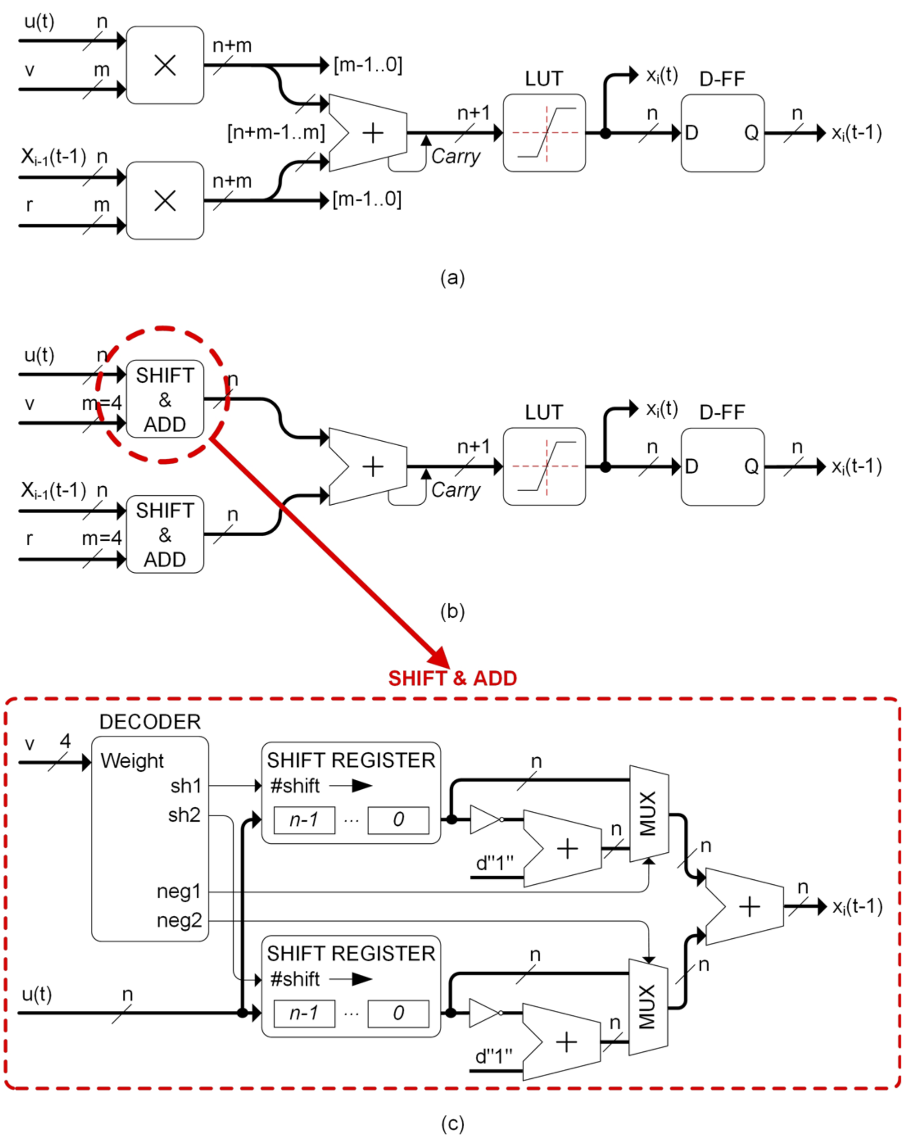

Fig. 3 (a-c) Neuron design: (a) general circuit design of the neuron; (b) reduced implementation scheme when the weight resolution is limited to few bits ($m = 4$) and the multipliers are replaced by simple shift-and-add blocks; (c) description of the shift-and-add block

of other nonlinear functions such as the step function or the piece-wise linear function. The Heaviside (or step) function represents the binary neuron, which corresponds to the first generation of neurons proposed by McCulloch and Pitts ([39]). In this case, the neuron can only give a digital output: it sends a binary high value ("1") if the sum of the weighted inputs surpasses

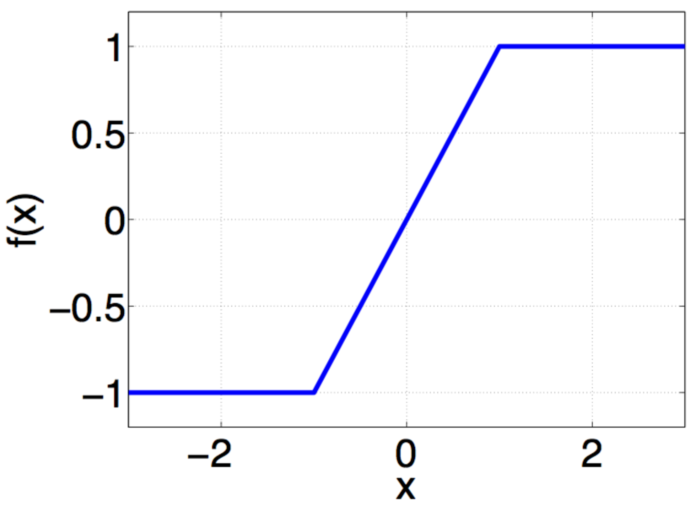

Fig. 4 Piece-wise function used as non-linear activation function.

the threshold level, and a low value (“0”) otherwise. On the other hand, the neurons using a continuous function (instead of the threshold one) belong to the second generation of neurons, which allows analog outputs. Networks of neurons of this type are more powerful than the ones based on first generation units (they can perform the same functions using fewer nodes). The neurons of the second generation are also more biologically realistic and similar to the spiking neurons than the first generation ones since they can model the spiking frequency (firing rate). The second-generation activation functions are often required to be continuous, derivable and bounded. The necessity for being derivable comes from the fact that the most common learning algorithms for training an ANN to perform a certain function need to compute the derivative of the transfer function [40]. For the special case of this work, in which the training is performed at the output layer and any back-propagation algorithm is implemented, a piece-wise linear function is able to provide very good fitting results along with a compact hardware implementation. In Fig.4 we show the non-linear function used for each internal neuron of the reservoir. The function is easily reproduced in hardware using a low gate count. As will be shown in the results’ section when comparing with other works, the use of a piece-wise linear function instead of a more sophisticated sigmoidal function is not compromising the system accuracy.

3.5 Benchmark Tasks

In time-series prediction or forecasting the objective is to predict future values based on previously observed ones. Thus, the input sequences are mapped onto a real-valued output sequence that represents one-step or several-step ahead predictions of the desired variable. That is, the value of the series at the current time is introduced each time step as input to the system and the time-series value corresponding to a given time horizon must be predicted. In this work we test the proposed RC methodology with respect to three widely used benchmarks in the RC literature that are the Mackey Glass [41] and Santa Fe [42] time series prediction tasks along with the equalization of a nonlinear channel [43]. The time series processing is divided in two steps:

- Part of the time-series is used for off-line training using a R script. The optimum v and r values are selected along with the output layer weights ($\boldsymbol{W_{out}}$). Then, the network is automatically generated using a hardware description language code (VHDL).
- The rest of the time-series is digitized to 16bits two's complement that is transferred to the on-chip RAM memory of the FPGA for its processing.

The performance of the time-series forecasting task is evaluated using the normalized mean square error (NMSE) and the Root Mean Square Error (RMSE):

$$NMSE = \frac{\sum_{i=1}^{L'} \left(y_i - \hat{y}_i\right)^2}{\sum_{i=1}^{L'} \left(y_i - \bar{y}\right)^2} \tag{8}$$

where $\boldsymbol{y} = \big(y(1), y(2), .., y(L')\big)$ is the time series to be predicted (target), $\hat{\boldsymbol{y}} = \big(\hat{y}(1), \hat{y}(2), .., \hat{y}(L')\big)$ is the predicted value provided by the output layer of the reservoir following equation (2), and $\bar{y}$ is the mean value of $\boldsymbol{y}$. Parameter L' is the number of samples in the test set. The RMSE is estimated as:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{L'} \left(y_i(t) - \hat{y_i}(t)\right)^2}{L'}} \tag{9}$$

Other error estimation also used in the context of this work is the Mean Absolute Error (MAE), defined as:

$$MAE = \frac{1}{L'} \sum_{i=1}^{L'} |y_i(t) - \hat{y_i}(t)| \tag{10}$$

Finally, the quality for the equalization process is measured as the fraction of symbols incorrectly classified with respect the total number of symbols (symbol error rate, SER):

$$SER = \frac{Number\, of\, misclassified\, values}{Total\, number\, of\, values} \tag{11}$$

3.5.1 The Mackey-Glass chaotic time series prediction task

The first benchmark used in this work is the time series prediction obtained from the chaotic Mackey-Glass system. Defining the delayed signal as $u_{t_D} \equiv u(t - t_D)$, the Mackey-Glass oscillator is defined by the following delay differential equation [41]:

$$\frac{du_0}{dt} = \frac{\beta u_\tau}{1 + u_\tau^p} - bu_0 \qquad (12)$$

where the typical values $\beta = 0.2$, $b = 0.1$, $p = 10$ and $\tau = 17$ have been selected in this work. In particular, we use a time series with 10.000 points obtained from $u(t)$ with a sampling of 3 time steps. The time series is normalized with zero mean, while the 60% of the time series is used for training and the remaining 40% for testing. The goal for the processing task is to predict the next sample in the chaotic time trace before it has been injected into the reservoir computer (for one-step ahead prediction). To evaluate the performance of the network, we use the NMSE defined in (8).

3.5.2 The Santa Fe prediction task

The Santa Fe laser time-series prediction task is a widely used benchmark in the RC literature [38]. The task consists in forecasting an experimental recording of the output power of a far-infrared laser operating in the chaotic regime, that is usually evaluated for one-step ahead predictions. A fragment of such time-series can be observed in Fig.5.

In this work, we employ a total of 4.000 samples of the original laser dataset, the first 75% for training and the remaining 25% for testing. The goal for the Santa-Fe task is to predict the next sample in the chaotic time trace before it has been injected into the reservoir computer (one-step ahead prediction). The performance of this task is evaluated using the normalized mean square error (NMSE) (8).

3.5.3 Nonlinear channel equalization

As a third task, we evaluate the performance of the system for the equalization of a wireless communication channel. Communication systems are aimed at efficiently sending information from a transmitter to a receiver using an available channel. This requires a processing of the received data as the channel is always responsible for distorting to some degree the transmitted signals [44]. The nature of the modifications suffered by the data sent depends on the particular features of the channel model, which can be either linear or nonlinear. In the case of satellite and mobile-phone communications, the sender's power amplifier must work in the high-gain region, close to the saturation point, in order to ensure a low power. This fact adds important nonlinear distortions in the communication channel, which may be compensated either at the transmitter side with pre-distortion or at the receiver with equalization. Here, we focus on the digital compensation of the nonlinear channel at the receiver side.

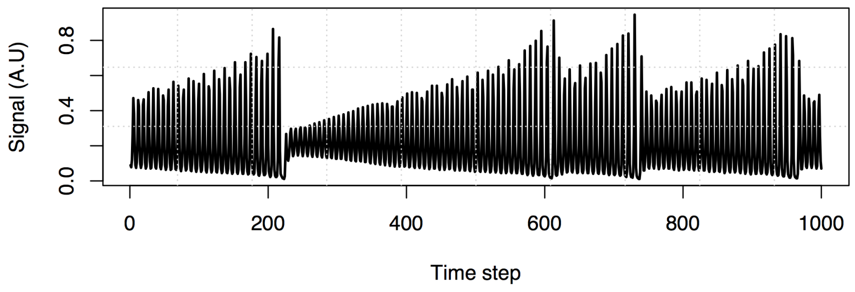

Fig. 5 The Santa Fe laser time-series presents different increasing oscillations and abrupt amplitude changes that make it optimum for testing different prediction methodologies

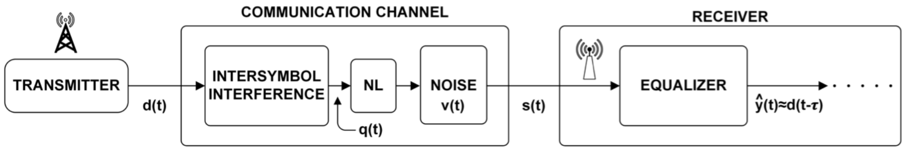

Fig. 6 Schematic diagram of a wireless communication system with a channel equalizer

To sum up, the problem of channel equalization consists of designing a device (the equalizer) that is present in the receiver and is intended to cancel the distortions introduced by the physical environment used for transmission, thus enabling the correct recovery of the original information [45].

The schematic of a communication system using a channel equalizer is illustrated in Fig.6. The transmitter communicates the symbol sequence $d(t)$ as an analog envelope signal modulated on a high-frequency carrier signal. Then, it is received and demodulated into the analog signal $s(t)$, which is a corrupted version of $d(t)$. The main sources of corruption are the linear superposition of adjacent symbols ($q(t)$, inter-symbol interference), nonlinear distortion induced by operating the sender's power amplifier in the high-gain region, and noise ($\nu(t)$, thermal or due to interfering signals). The corrupted signal $s(t)$ is then passed through the equalizer with the purpose of recovering the transmitted sequence $d(t)$ or its delayed version $d(t-\tau)$, where τ represents here the propagation delay associated with the physical channel. The equalized signal $\hat{y}(t)$ is finally converted back into a symbol sequence $(\hat{d}(t-\tau))$. The function performed by the equalizer is adapted during the training stage so that the reservoir output $\hat{y}(t)$ can restore $s(t)$ to $d(t-\tau)$ as closely as possible. Using the training data ($s(t)$ as input and $d(t-\tau)$ as desired output), the equalizer parameters (weights) are adjusted so as to minimize the error $e(t)$, defined as the difference between the target output $d(t-\tau)$ and the equalization output $\hat{y}(t)$: $e(t) \equiv d(t-\tau) - \hat{y}(t)$. Once the training has been completed, the equalizer weights are fixed and used to estimate the transmitted sequence. Linear filters have been widely used as equalizers due to their simplicity and mathematical tractability [46, 47]. However, their performance is not satisfactory for highly nonlinear and dispersive channels. Channel equalizers based on more complex approaches, such as polynomial filters (using Volterra series expansions, [48, 49, 50]) and artificial neural networks (ANN) [51, 52, 53], were developed showing a higher performance than the linear channel equalizers. Among ANN models, echo state networks (ESNs) represent an attractive equalization solution since they are both nonlinear and recurrent, which makes possible to meet the memory and mapping requirements of this particular task with the advantage of not posing complex optimization problems. The ESN approach has been proposed and investigated for the nonlinear channel equalization task using channel models of diverse complexity [38, 45, 54, 55]. The results presented in [55] show that the ESN approach is able to reach the same performance as a state-of-the-art Volterra equalizer while presenting similar complexity. Here, we propose and analyze the use of a digital hardware implementation of the ESN to perform the equalization of a wireless communication channel. A low-cost hardware implementation of the channel equalizer is of great interest in wireless communications given the limited available power in mobile phone devices and aboard satellites. To create the data-set, we have taken the channel model of a nonlinear wireless transmission system from [56]. This model only considers real inputs. A more realistic extension making use of complex symbols can be found in [57]. The input to the channel is an independent and identically distributed random sequence $d(t)$ with values selected from $\{-3, -1, 1, 3\}$. Then, $d(t)$ values are used to form a sequence $q(t)$ through a linear filter as follows:

$$\begin{aligned} q(t) = {} & 0.08d(t+2) - 0.12d(t+1) + d(t) + 0.18d(t-1) \\ & - 0.1d(t-2) + 0.09d(t-3) - 0.05d(t-4) \\ & + 0.04d(t-5) + 0.03d(t-6) + 0.01d(t-7) \end{aligned} \tag{13}$$

Finally, a noisy nonlinear transformation is applied to generate s(t):

$$s(t) = q(t) + 0.036q(t)^2 - 0.011q(t)^3 + \nu(t) \tag{14}$$

where $\nu(t)$ is an independent and identically distributed Gaussian noise with zero mean. The equalization task consists in getting the output $y(t) = d(t-2)$ when the corrupted signal $s(t)$ is presented at the network input. The equalized signal $\hat{y}(t)$ needs to be finally converted back into a 4-symbol sequence. This is done by equidistant thresholding. That is to say, the symbol $+3$ is chosen if $\hat{y}(t) > 2$, $+1$ if $2 \geq \hat{y}(t) > 0$, etc. Therefore, the equalization task is actually a classification problem where deviations of the estimated output with respect to the target signal are acceptable as long as the values are kept within the correct classification boundaries. As in the previous tests, first the reservoir is trained off-line with a training set of 6.000 points. Once the network is trained, a VHDL file is generated to test the network. A test set of 4.000 points is digitized to 16bits two's complement that is inserted in the on-chip RAM memory of the FPGA for its processing.

4 Results

The proposed methodology is synthesized using echo state networks with cyclic topology (SCR) (Fig.1b), encoded using VHDL on an ALTERA Cyclone IV FPGA. The VHDL code used is composed of three parts: RAM memory (containing the input to be processed), the reservoir (constructed using the programmable logic elements of the FPGA), and the output layer (using the dedicated multipliers of the FPGA). We compare the performance of the proposed model with respect to different widely used benchmark tasks and comparing with some previously published works. Regarding the area impact of the proposed methodology, it represents an 86% of area reduction if compared with the standard digital realization. In Fig.7 we show the number of logic elements used by the proposed methodology in comparison with a conventional digital solution when implemented in an ALTERA FPGA device (Cyclone IV EP4CE115F297C7N).

The training of the system is performed following the standard procedure for ESNs [17], which consists in a linear regression of the desired output on the reservoir states. A convenient numerical model of the hardware reservoir is employed for extracting the output weights $\boldsymbol{W_{out}}$ during the training phase. These weights are used in the training set to determine the optimum internal configuration parameters (values of r and v). This is done scanning off-line for all the possible network configurations. Then, the hardware is set up with the obtained optimum configuration parameters and weight matrix ($\boldsymbol{W_{out}}$) and the final error is evaluated using the test set. Fig.8 shows how the network performance (NMSE) varies with v and r for the special case of the Santa-Fe time-series forecasting problem). This graph is explored to establish the optimum internal setup. It can be observed (Fig.8) that the discrete weight values allowed by the "multiplier-less" approach (constrained in the range from 0 to 1 with steps of 0.125 and highlighted with asterisks in Fig.8) ensure errors that are in close proximity to the best result provided by the numerical simulations. Once the optimum parameters are obtained ($r = 0.875$ and $v = 1$ in the case of Fig.8), the FPGA is configured and the test set is stored into an internal RAM memory of the FPGA. The memory is used to provide each new input value to the reservoir every time step (each N clock cycles) and the resulting outputs (individual neuron states) are processed by the FPGA, thus generating the output $\hat{\boldsymbol{y}}$. This computation is performed following equation (2) in a total of N clock cycles so that the processing speed of the proposed design is f/N, where f is the clock frequency fixed to $50MHz$. This output is extracted from the FPGA with a logic analyzer and used to calculate the system's performance as the error between the estimated and targeted values. The main drawback of the proposed approach is that the internal configuration is restricted to some discrete values and that the optimum internal weights cannot be selected. Nevertheless, this limitation has not impacted too much the accuracy that is maintained within competitive values (as will be shown in the results' section). To illustrate all this process, in Fig.9 we represent graphically

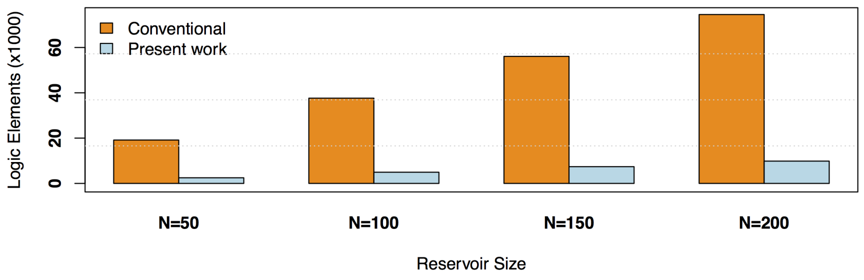

Fig. 7 Spent hardware resources of the Cyclone IV (EP4CE115F297C7N) FPGA for different sizes of the reservoir (N), using a conventional solution and the proposed design

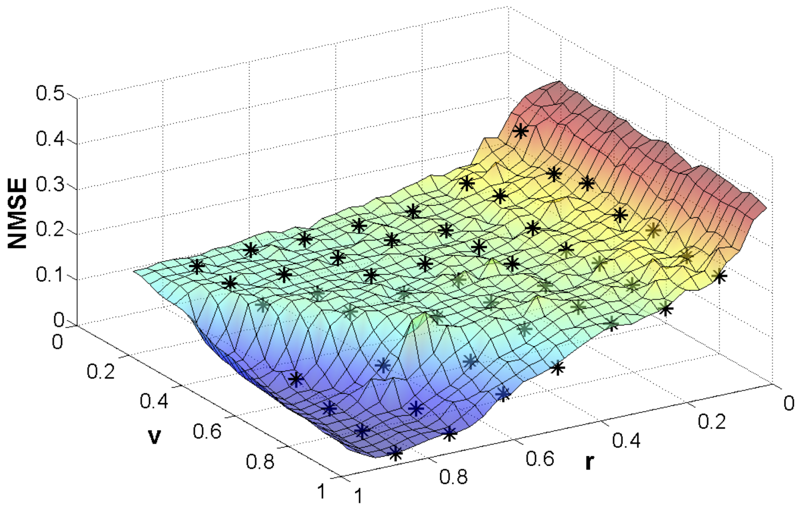

Fig. 8 Performance (NMSE) of the RC implementation (with $N = 200$ neurons) in the Santa-Fe time-series prediction task as a function of the weight parameters r and v. The surface is obtained numerically while the highlighted points represent the possible discrete values to which the multiplier-less approach is limited

the data processing flowchart realized for the experimental validation of the model.

4.1 The Mackey-Glass oscillator

We first consider the measurements obtained using the popular example of time series prediction with the chaotic Mackey-Glass system. We process a

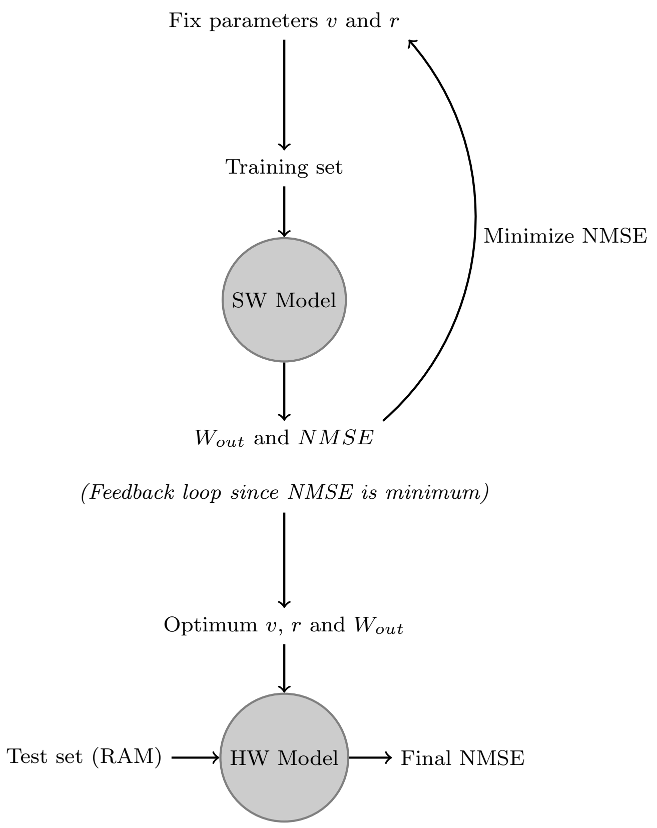
Fix parameters v and r
Training set
SW Model
Minimize NMSE
W_{out} and $NMSE$
(Feedback loop since NMSE is minimum)
Optimum v, r and W_{out}
Test set (RAM)
HW Model
Final NMSE

Fig. 9 Flowchart followed for testing the network model

discrete time series composed of 10.000 points obtained from the continuous system (12) with sampling time of 3 time steps. The time series is divided into two sets, where the first 6.000 points are used for training and the next 4.000 for testing. The network parameters that optimize the output fitting are ($r = 0.875$ and $v = 1$) . In Fig.10 we show the first 150 iterations of the test set (measurements $\hat{y}(t)$ are represented with a solid line and $u(t)$ with circles) using OLS. As can be appreciated, a good approximation is obtained between the model and the desired output.

We estimate the residuals $\boldsymbol{e} = \boldsymbol{y} - \hat{\boldsymbol{y}}$ of the fitting for the training set to evaluate the possible heteroscedasticity of data. In Fig. (11) we observe that residuals present an apparent homogeneity in their distribution. We compared also the linear fitting obtained by using an Ordinary Least Squares (OLS) approximation using expression (3) and also Feasible Generalized Least Squares (FGLS) using expression (6). For the calculation of the residual's covariance matrix $\boldsymbol{\Omega}$ we used the estimations proposed in [58, 59]. In Table 1 we compare the fitting (the Mean Absolute Error) of the two methodologies, showing that FGLS is unable to provide a better fitting when compared to OLS. This

Table 1 Performance results of the Mackey Glass chaotic oscillator for OLS and FGLS fitting methods (N=300)

Method	Training error (MAE)	Test error (MAE)
OLS	0.00186	0.00198
FGLS[58]	0.0019	0.00206
FGLS[59]	0.0019	0.00204

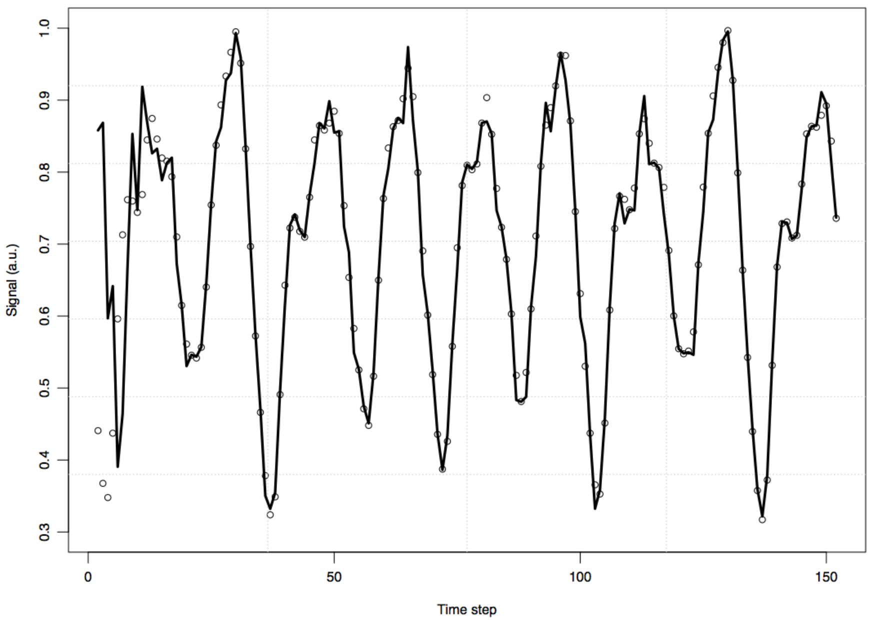

Fig. 10 FPGA measurements showing the first 150 iterations of the test set ($\hat{y}(t)$ with a solid line and $u(t)$ with circles)

fact can be due to the intrinsic symmetry of the reservoir. Therefore, in this work, the estimation of the output layer weights W_{out} is performed using the Ordinary Least Squares method for simplicity.

The error results obtained for the full test set are provided in Table 2, where we compare the measurements taken with the proposed ESN model and the previously-published works [26, 29] that implements different networks with a different number of neurons and using an experimental optoelectronic setting. As can be appreciated, the proposed network model is able to provide a performance that is similar to those experimental implementations. The comparison is done through the estimation of the NMSE and the RMSE.

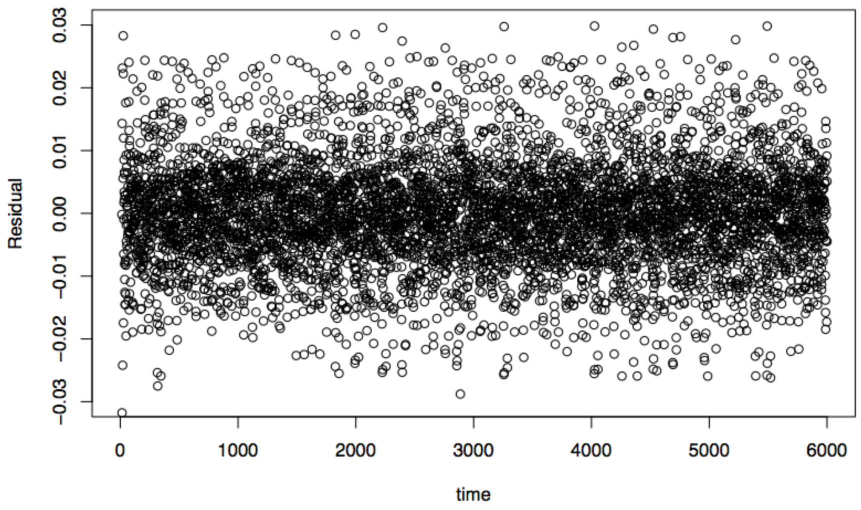

Fig. 11 Variation of residual (e) with time step for the Mackey-Glass oscillator. An apparent homoscedasticity of signals can be appreciated

Table 2 Performance results of the Mackey Glass chaotic oscillator for the proposed design and previously published models based on Echo State Networks (ESN) and an Extreme Learning Machine (ELM)

Technology	Method	Ref.	N	NMSE	$log_{10}(RMSE)$	Type
Optoelectronic	ELM	[26]	800	-	~ -1.95	Hard.
Optoelectronic	ESN	[26]	~ 615	-	~ -1.24	Hard.
Optoelectronic	RNN	[29]	900	0.013	-	Hard.
FPGA	ESN	This work	300	0.010	-1.74	Hard.

4.2 The Santa Fe prediction task

The performance of the proposed system is also tested for the Santa Fe time-series prediction task [42]. Networks with different sizes (for 48 and 200 neurons) are implemented and analyzed using the proposed "multiplier-less" approach (Fig.3b) with a precision of 16 bits ($n = 16$). For this task we employ 4.000 samples of the original laser data-set, the first 3.000 for training and the remaining 1.000 for testing.

Fig.12 illustrates the experimental measurements obtained through the proposed design when using 48 and 200 neurons. As can be observed, the fitting of the measurements taken in the FPGA are improved when increasing N. The performance of the network as N increases can be observed visually in Fig.13, where the expected values are plotted vs. the FPGA measurements.

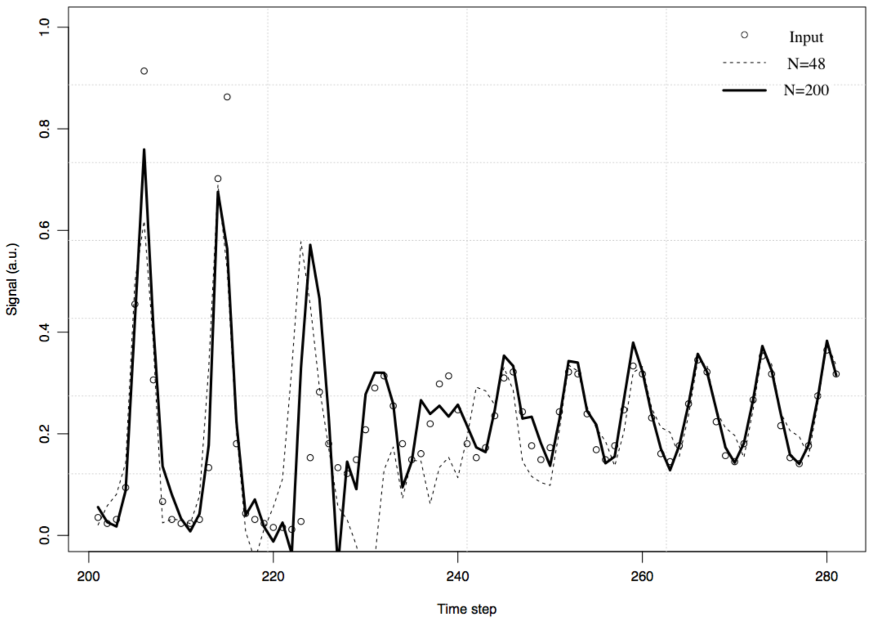

Fig. 12 Fragment of the Santa Fe time-series test set: original values and one-step ahead predictions performed by the proposed reservoir implementation with $N = 48$ and $N = 200$ neurons. As can be appreciated, the network is able to adapt to the abrupt changes of the input

As can be appreciated from the measurements, the fitness improves as N increases.

In Table 3 we show the performance of the proposed model measured in terms of the NMSE, speed (in points predicted per second) and power dissipation for the processing of the Santa-Fe time series prediction task. Comparison with previously published models is also included in the table. We distinguish between theoretical values of NMSE obtained from high-precision numerical calculations (simulation) and NMSE values obtained from experimental settings (Hard.). These two values can differ significantly due to the intrinsic complexity of experimental settings that may present both system and quantization noise. This is evidenced in reference [28], where the expected NMSE provided by software is considerably lower than the measured one. We also show the results of different studies that are purely numerical as the work in [38] showing the expected performance of Simple Cycle Reservoir designs, or the paper in [43] where a semiconductor ring laser with optical feedback (SRL) is numerically simulated. We also provide in the table the power-delay product (PDP) achieved by the experimental settings (a classical figure of merit of the overall hardware performance). As can be appreciated the proposed design is

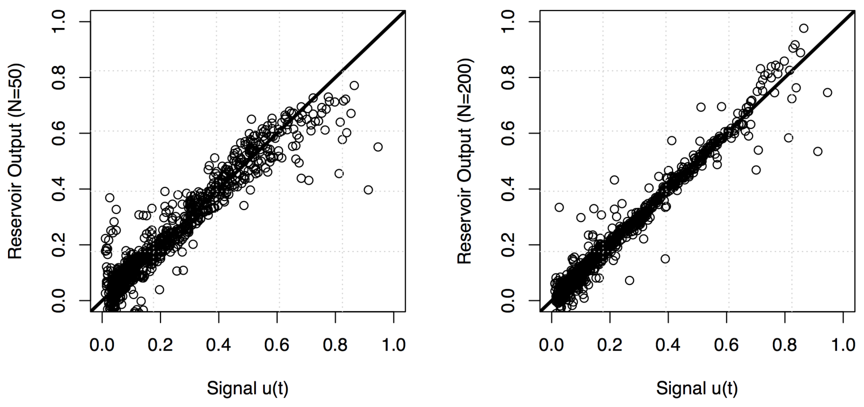

Fig. 13 Expected vs. estimated values provided by the FPGA for the 1000 points of the test set. A considerable better fitness is obtained for $N = 200$

Table 3 Performance results of the Santa Fe experiments for the proposed design and some previously published models

Technology	Ref.	N	NMSE (Hard.)	NMSE (Simulation)	speed (pps)	Power (W)	PDP ($W \cdot \mu s$)
Optoelectronic	[43]	200	-	0.02	-	-	-
Optoelectronic	[27]	388	0.106	-	$1.3 \cdot 10^7$	150	11.5
Optoelectronic	[28]	400	-	0.021	-	-	-
Numerical	[38]	200	-	0.008	-	-	-
Numerical	[38]	50	-	0.018	-	-	-
Analog Circuit	[25]	400	0.031	-	-	-	-
FPGA	[24]	50	0.131	-	1142	0.083	72.6
FPGA	[23](16b)	50	0.075	-	763	< 1.5	1966
FPGA	[23](12b)	50	0.12	-	12207	< 1.5	123
FPGA	This work	200	0.079	0.0766	$2.5 \cdot 10^5$	< 1.5	6
FPGA	This work	48	0.148	0.144	10^6	< 1.5	1.5

able to provide the lowest PDP that can be understood as the mean energy needed per operation [27, 28].

4.3 Channel equalization

The nonlinear channel equalization task tries to recover an input symbol sequence from the signal received at the output of a standardized nonlinear noisy multipath RF (Radio-Frequency) channel. The performance is evaluated in terms of symbol error rate (SER), which is the fraction of misclassified symbols 11. Our results were obtained using a sequence of 6.000 training symbols and 4.000 test symbols with $N = 27$. To illustrate the network operation,

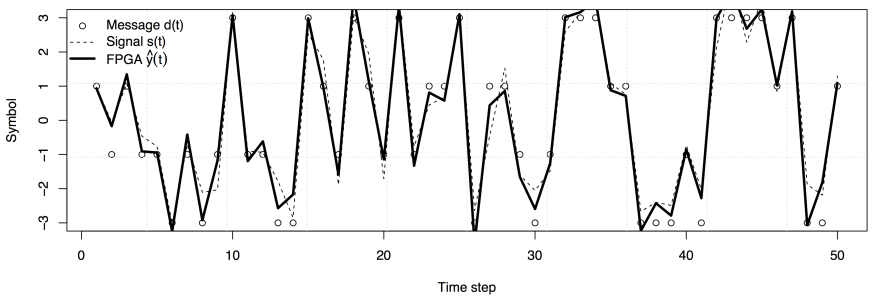

Fig. 14 Results of the channel equalization problem, the expected signal $d(t)$ is represented with circles and the received signal $s(t)$ is shown with a dashed line. The corrected signal provided by the reservoir $\hat{y}(t)$ is represented with a solid line

Table 4 Results of the equalizer experiments for the proposed design and some previously published models

Technology	Ref.	N	SER (16dB)	SER (20dB)	SER (24dB)	SER (28dB)	Type
(Untreated)			0.167	0.131	0.114	0.098	
FPGA	This work	27	0.0345	0.0095	0.007	0.0042	Hard.
Optoelectronic	[26]	246	0.05	0.013	0.007	0.0025	Hard.
Optoelectronic	[60]	50	0.025	0.006	0.0007	0.00055	Sim.

in Fig.14 we show a total of 50 points taken from the test set. The expected signal $d(t)$ is represented with circles, the received signal $s(t)$ is shown with a dashed line and the measurements provided by the FPGA $\hat{y}(t)$ are shown with a black solid line. Some $s(t)$ points would be misclassified (as points 27 or 30) and the corrected signal provided by the reservoir ($\hat{y}(t)$, solid line) is able to provide the correct result. The overall results of the test set are shown in Table 4, where we estimate the Symbol Error Rate (SER) for four different signal to noise ratio values ($SNR = \{16dB, 20dB, 24dB, 28dB\}$). As can be appreciated, the proposed design is able to decrease considerably the SER with respect the untreated data. These SER values are also very similar with other hardware solutions based on the use of optoelectronic devices [26]. As done in Table 3, we compare the obtained results with both measurements taken from experimental settings (denoted as Hard. in the table) and purely numerical studies (simulation). Note that the results provided in [60] do not take into account the intrinsic problems (as system or quantization noise) that is present in experimental settings as in [26].

5 Conclusions

In this work, we have presented a fully parallel and efficient digital implementation of reservoir computing systems. The hardware realization is based on the observation that the reservoir connection weights can be limited to a few discrete values without compromising the system's performance. This can be conveniently exploited setting the synapses' weights to powers of two and sums of powers of two, which allow performing the multiplications with shift-and-add blocks, thus consuming minimal area. The validity of the resulting implementation has been demonstrated for different benchmarks: the Santa Fe and Mackey Glass oscillator time-series prediction tasks, and the equalization of a nonlinear channel. Performance comparisons with ten previous works that use the network scheme of Reservoir Computing systems are shown. The comparisons show that the proposed model represent a considerable improvement in terms of speed and power dissipation while is able to provide a similar accuracy than previous models. The proposed design enables the implementation of large reservoir networks using few hardware resources extending the range of efficient implementations of neural circuits in digital hardware [1, 2, 15, 16]. Considering the accuracy of the model, processing speed, along with the total power consumption, the proposed implementation represent an advantage with respect other RC circuit implementations (analog, digital and optoelectronic) due in part to the use of a fully parallel-connected neural network. To the best of our knowledge, the proposed design provide the lowest performance in terms of power-delay metrics so this technique is the most qualified to support portable real-time signal processing applications such as image and video sequence classification, which often require timely responses with a low power cost [19]. At the same time the proposed approach can be useful to perform specialized systems implementing computational intelligence techniques and requiring a low power consumption. Other potential applications to which this approach is more suitable could be speech recognition [10], robotics [18], wireless sensor networks [61], predictive controllers [7] and the classification of medical signals [9] among others. To sum up, it has been shown that the use of low-resolution weights for the reservoir has little effect on the system's performance while it allows a considerable reduction of the hardware and power resources. This observation makes possible a very compact implementation of massive reservoir networks with parallel processing capabilities.

References

1. Baptista D, Abreu S, Freitas F, Vasconcelos R, Morgado-Dias F (2013) A survey of software and hardware use in artificial neural networks. Neural Computing and Applications 23(3-4):591–599
2. Misra J, Saha I (2010) Artificial neural networks in hardware: A survey of two decades of progress. Neurocomputing 74(1-3):239–255

3. Baptista FD, Morgado-Dias F (2017) Automatic general-purpose neural hardware generator. Neural Computing and Applications 28(1):25–36
4. Amir MF, Kim D, Kung J, Lie D, Yalamanchili S, Mukhopadhyay S (2017) NeuroSensor: A 3D image sensor with integrated neural accelerator. In: 2016 SOI-3D-Subthreshold Microelectronics Technology Unified Conference, S3S 2016
5. Krizhevsky A, Sutskever I, Geoffrey E H (2012) ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems 25 (NIPS2012) pp 1–9
6. Morro A, Canals V, Oliver A, Alomar ML, Galan-Prado F, Ballester PJ, Rossello JL (2017) A Stochastic Spiking Neural Network for Virtual Screening
7. Li H, Zhang D, Foo SY (2006) A stochastic digital implementation of a neural network controller for small wind turbine systems. IEEE Transactions on Power Electronics 21(5):1502–1507
8. Chauhan A, Semwal S, Chawhan R (2013) Artificial neural network-based forest fire detection system using wireless sensor network. 2013 Annual IEEE India Conference (INDICON) pp 1–6
9. Raghunathan S, Gupta SK, Ward MP, Worth RM, Roy K, Irazoqui PP (2009) The design and hardware implementation of a low-power real-time seizure detection algorithm. Journal of Neural Engineering 6(5):056,005
10. Lee M, Hwang K, Park J, Choi S, Shin S, Sung W (2016) FPGA-based low-power speech recognition with recurrent neural networks. In: IEEE Workshop on Signal Processing Systems, SiPS: Design and Implementation, pp 230–235
11. Basterretxea K, Tarela JM, del Campo I (2002) Digital design of sigmoid approximator for artificial neural networks. Electronics Letters 38(1):35–37
12. Baptista D, Morgado-Dias F (2013) Low-resource hardware implementation of the hyperbolic tangent for artificial neural networks. Neural Computing and Applications 23(3-4):601–607
13. Nascimento I, Jardim R, Morgado-Dias F (2013) A new solution to the hyperbolic tangent implementation in hardware: Polynomial modeling of the fractional exponential part. Neural Computing and Applications 23(2):363–369
14. Carrasco-Robles M, Serrano L (2009) Accurate differential tanh(nx) implementation. International Journal of Circuit Theory and Applications 37(5):613–629
15. Nedjah N, De MacEdo Mourelle L (2007) Reconfigurable hardware for neural networks: Binary versus stochastic. Neural Computing and Applications 16(3):249–255
16. Lotrič U, Bulić P (2012) Applicability of approximate multipliers in hardware neural networks. Neurocomputing 96:57–65
17. Lukoševičius M, Jaeger H, Schrauwen B (2012) Reservoir Computing Trends. KI - Künstliche Intelligenz 26(4):365–371

18. Antonelo EA, Schrauwen B (2015) On Learning Navigation Behaviors for Small Mobile Robots with Reservoir Computing Architectures. IEEE Transactions on Neural Networks and Learning Systems 26(4):763–780
19. Jalalvand A, Wallendael GV, Walle RVD (2015) Real-Time Reservoir Computing Network-Based Systems for Detection Tasks on Visual Contents. In: Proceedings - 7th International Conference on Computational Intelligence, Communication Systems and Networks, CICSyN 2015, pp 146–151
20. Bacciu D, Barsocchi P, Chessa S, Gallicchio C, Micheli A (2014) An experimental characterization of reservoir computing in ambient assisted living applications. Neural Computing and Applications 24(6):1451–1464
21. Lin X, Yang Z, Song Y (2009) Short-term stock price prediction based on echo state networks. Expert Systems with Applications 36(3 PART 2):7313–7317
22. Buteneers P, Verstraeten D, Nieuwenhuyse BV, Stroobandt D, Raedt R, Vonck K, Boon P, Schrauwen B (2013) Real-time detection of epileptic seizures in animal models using reservoir computing. Epilepsy Research 103(2-3):124–134
23. Alomar ML, Canals V, Perez-Mora N, Martínez-Moll V, Rosselló JL (2016) FPGA-based stochastic echo state networks for time-series forecasting. Computational Intelligence and Neuroscience 2016
24. Alomar ML, Soriano MC, Escalona-Morán M, Canals V, Fischer I, Mirasso CR, Rosselló JL (2015) Digital Implementation of a Single Dynamical Node Reservoir Computer. IEEE Transactions on Circuits and Systems II: Express Briefs 62(10):977–981
25. Soriano MC, Ortín S, Keuninckx L, Appeltant L, Danckaert J, Pesquera L, van der Sande G (2015) Delay-Based Reservoir Computing: Noise Effects in a Combined Analog and Digital Implementation. IEEE Transactions on Neural Networks and Learning Systems 26(2):388–393, DOI 10.1109/TNNLS.2014.2311855
26. Ortín S, Soriano MC, Pesquera L, Brunner D, San-Martín D, Fischer I, Mirasso CR, Gutiérrez JM (2015) A Unified Framework for Reservoir Computing and Extreme Learning Machines based on a Single Time-delayed Neuron. Scientific Reports 5
27. Brunner D, Soriano MC, Mirasso CR, Fischer I (2013) Parallel photonic information processing at gigabyte per second data rates using transient states. Nature Communications 4
28. Hicke K, Escalona-Morán M, Brunner D, Soriano MC, Fischer I, Mirasso CR (2013) Information Processing Using Transient Dynamics of Semiconductor Lasers Subject to Delayed Feedback. IEEE Journal of Selected Topics in Quantum Electronics 19(4):1501,610–1501,610
29. Bueno J, Maktoobi S, Froehly L, Fischer I, Jacquot M, Larger L, Brunner D (2018) Reinforcement learning in a large-scale photonic recurrent neural network. Optica 5(6):756–760
30. Liu Y, Nie L, Han L, Zhang L, Rosenblum DS (2015) Action2Activity: Recognizing complex activities from sensor data. In: IJCAI International

Joint Conference on Artificial Intelligence, vol 2015-January, pp 1617–1623

31. Liu Y, Zhang L, Nie L, Yan Y, Rosenblum DS (2016) Fortune Teller: Predicting Your Career Path. Proceedings of the 30th Conference on Artificial Intelligence (AAAI 2016) (1):201–207
32. Liu Y, Nie L, Liu L, Rosenblum DS (2016) From action to activity: Sensor-based activity recognition. Neurocomputing 181:108–115
33. Draper NR, Smith H (1998) Applied Regression Analysis. Technometrics 47(3):706
34. Chatterjee S, Hadi AS (1986) Influential Observations, High Leverage Points, and Outliers in Linear Regression. Statistical Science 1(3):379–393
35. Boukouvalas A, Cornford D, Stehlík M (2014) Optimal design for correlated processes with input-dependent noise. Computational Statistics and Data Analysis 71:1088–1102
36. Lim YC, Liu B (1988) Design of Cascade Form FIR Filters with Discrete Valued Coefficients. IEEE Transactions on Acoustics, Speech, and Signal Processing 36(11):1735–1739
37. Marchesi M, Orlandi G, Piazza F, Uncini A (1993) Fast Neural Networks Without Multipliers. IEEE Transactions on Neural Networks 4(1):53–62
38. Rodan A, Tiňo P (2011) Minimum complexity echo state network. IEEE Transactions on Neural Networks 22(1):131–144
39. McCulloch WS, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. The Bulletin of Mathematical Biophysics 5(4):115–133
40. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323(6088):533–536
41. Mackey MC, Glass L (1977) Oscillation and chaos in physiological control systems. Science (New York, NY) 197(4300):287–9
42. Weigend AS, Gershenfeld NA (1993) Results of the time series prediction competition at the Santa Fe Institute. In: IEEE International Conference on Neural Networks - Conference Proceedings, vol 1993-Janua, pp 1786–1793
43. Modeste Nguimdo R, Verschaffelt G, Danckaert J, Van Der Sande G (2015) Simultaneous computation of two independent tasks using reservoir computing based on a single photonic nonlinear node with optical feedback. IEEE Transactions on Neural Networks and Learning Systems 26(12):3301–3307
44. Benedetto S, Biglieri E (1999) Principles of Digital Transmission: With Wireless Applications. Kluwer Academic Publishers, Norwell, MA, USA
45. Boccato L, Lopes A, Attux R, Von Zuben FJ (2011) An echo state network architecture based on volterra filtering and PCA with application to the channel equalization problem. In: Proceedings of the International Joint Conference on Neural Networks, pp 580–587
46. Lucky RW (1965) Automatic Equalization for Digital Communication. Bell System Technical Journal 44(4):547–588
47. Gersho A, Lim TL (1981) Adaptive Cancellation of Intersymbol Interference for Data Transmission. Bell System Technical Journal 60(9):1997–2021

48. Karam G, Sari H (1989) Analysis of Predistortion, Equalization, and ISI Cancellation Techniques in Digital Radio Systems with Nonlinear Transmit Amplifiers. IEEE Transactions on Communications 37(12):1245–1253
49. Mathews VJ (1991) Adaptive Polynomial Filters. IEEE Signal Processing Magazine 8(3):10–26
50. Malone J, Wickert MA (2011) Practical Volterra equalizers for wideband satellite communications with TWTA nonlinearities. In: 2011 Digital Signal Processing and Signal Processing Education Meeting, DSP/SPE 2011 - Proceedings, pp 48–53
51. Chen S, Gibson GJ, Cowan CFN (1990) Adaptive channel equalisation using a polynomial-perceptron structure. I Communications, Speech and Vision 137(5):257–264
52. Patra JC, Pal RN, Baliarsingh R, Panda G (1999) Nonlinear channel equalization for QAM signal constellation using artificial neural networks. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 29(2):262–271
53. Patra JC, Meher PK, Chakraborty G (2009) Nonlinear channel equalization for wireless communication systems using Legendre neural networks. Signal Processing 89(11):2251–2262
54. Jaeger H, Haas H (2004) Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication. Science 304(5667):78–80
55. Bauduin M, Smerieri A, Massar S, Horlin F (2015) Equalization of the non-linear satellite communication channel with an Echo state network. In: IEEE Vehicular Technology Conference, vol 2015
56. Mathews VJ, Lee J (1994) Adaptive algorithms for bilinear filtering. In: Proceedings of SPIE - The International Society for Optical Engineering, vol 2296, pp 317–327
57. Seth S, Ozturk MC, Principe JC (2007) Signal processing with echo state networks in the complex domain. In: Machine Learning for Signal Processing 17 - Proceedings of the 2007 IEEE Signal Processing Society Workshop, MLSP, pp 408–412
58. Cribari-Neto F (2004) Asymptotic inference under heteroskedasticity of unknown form. Computational Statistics and Data Analysis 45(2):215–233
59. White H (1980) A heteroskedasticity-consistent covariance matrix and a direct test for heteroskedasticity. Econometrica 48:817–838
60. Vinckier Q, Duport F, Smerieri A, Haelterman M, Massar S (2016) Autonomous bio-inspired photonic processor based on reservoir computing paradigm. In: 2016 IEEE Photonics Society Summer Topical Meeting Series, SUM 2016, pp 183–184
61. Mathews E, Poigné A (2008) An Echo State Network based pedestrian counting system using wireless sensor networks. In: 2008 International Workshop on Intelligent Solutions in Embedded Systems (WISES 2008), pp 1–14