Neural Process Lett (2017) 46:899–914
DOI 10.1007/s11063-017-9655-x

FPGA Implementation of Neurocomputational Models: Comparison Between Standard Back-Propagation and C-Mantec Constructive Algorithm

Francisco Ortega-Zamorano[1] · José M. Jerez[2] · Gustavo E. Juárez[3] · Leonardo Franco[2]

Published online: 16 June 2017

Abstract Recent advances in FPGA technology have permitted the implementation of neurocomputational models, making them an interesting alternative to standard PCs in order to speed up the computations involved taking advantage of the intrinsic FPGA parallelism. In this work, we analyse and compare the FPGA implementation of two neural network learning algorithms: the standard and well known Back-Propagation algorithm and C-Mantec, a constructive neural network algorithm that generates compact one hidden layer architectures with good predictive capabilities. One of the main differences between both algorithms is the fact that while Back-Propagation needs a predefined architecture, C-Mantec constructs its network while learning the input patterns. Several aspects of the FPGA implementation of both algorithms are analyzed, focusing in features like logic and memory resources needed, transfer function implementation, computation time, etc. The advantages and disadvantages of both methods in relationship to their hardware implementations are discussed.

Keywords Constructive neural networks · FPGA · Hardware implementation

✉ Leonardo Franco
lfranco@lcc.uma.es

Francisco Ortega-Zamorano
fortega@Yachaytech.edu.ec

José M. Jerez
jja@lcc.uma.es

Gustavo E. Juárez
gjuarez@herrera.unt.edu.ar

1 School of Mathematics and Computer Science, University of Yachay Tech, San Miguel de Urcuquí, Ecuador

2 Computer Science Department, ETSI Informática, Universidad de Málaga, Málaga, Spain

3 Facultad de Ciencias Exactas y Tecnología, Universidad Nacional de Tucumán, San Miguel de Tucumán, Argentina

1 Introduction

Artificial Neural Networks (ANN) [14,18,30] are mathematical models inspired in the functioning of the brain that can be utilized in clustering and classification problems, and that have been successfully applied in several fields, including pattern recognition, stock market prediction, control tasks, medical diagnosis and prognosis, industrial applications, etc. [4,13,15,19]. Neural Networks architectures, whether organized in layers or not, usually comprise several neurons having a large number of interconnections. Even if traditional PCs do not seem to be the most suitable technology for implementing an essentially parallel system, they have been the most used choice given the success of PC based computation and the possibility of also using clusters of PCs [12,33,38]. Alternative technologies like FPGAs, DSPs, custom VLSI chips, GPUs and microcontrollers have been utilized more recently at a larger scale, and in particular we can mention the recent widely popular utilization of GPUs for Deep Learning, a new and promising technology that is growing extremely fast due to its impressive prediction capabilities [7,22]. Nevertheless, in this work we will focus on FPGA technology, as the capacity and performance of current FPGAs are a realistic alternative for real time implementation of ANN. FPGAs are reprogrammable silicon chips, using prebuilt logic blocks and programmable routing resources, that can be configured to implement custom hardware functionality, being able also to change almost instantly its behaviour by recompiling a new circuitry configuration [3,5,16]. Recent advances in technology have permitted to construct FPGAs with considerable large amounts of processing power and memory storage, and as such they have been applied in several domains (Telecommunications, Robotics, Pattern recognition tasks, Infrastructure monitoring, etc.) [1,6,21]. In particular FPGAs seem quite suitable for Neural Network implementations as they can be programmed to operate in a parallel way [11,17,23,24,28,35,36,39].

Within the area of supervised pattern recognition, the efficient hardware implementation of neurocomputational models can be done mainly following two different strategies: modifying and adapting the traditional Back-Propagation (BP) algorithm or developing new algorithms better suited to the hardware constraints. In this work these two different possibilities are explored, first by doing a hardware optimization of the standard Back-Propagation algorithm [28], and secondly through the implementation of an alternative algorithm named Competitive Majority Network Trained by Error Correction (C-Mantec) based on an incremental constructive architecture [25]. Constructive Neural Networks algorithm (CoNN) are alternative models to the standard BP algorithm that generate the network topology on-line during the training phase, avoiding the complex problem of selecting an adequate neural architecture [8]. The overall idea of the work is to do a comparative analysis of the two approaches in order to identify the advantages and disadvantages for each case, helping with this information to the decision problem of choosing a neurocomputational model for the specific application. The Back-Propagation algorithm (BP) is the standard learning procedure for training multilayer neural networks architectures [31,37] but one of the main problems associated to its implementation is the lack of a clear methodology for determining the network topology before training starts [10]. On the other hand, C-Mantec [34] is a neural network constructive algorithm that utilizes competition between the neurons and a modified perceptron learning rule (thermal perceptron [9]) to build single hidden layer compact architectures with good prediction capabilities for the supervised classification problems. The novelty of C-Mantec in comparison to previous proposed constructive algorithms is that the neurons in the single hidden layer compete for learning the incoming data, and this process permits

Table 1 PC versus FPGA comparison

	FPGA	CPU-PC
Model	Virtex-5 XC5VLX110T	Intel Core i5 3 GHZ
Price	∼750 $	∼650 $
Consumption	∼50 W	∼160 W
BP algorithm execution time	0.19 ± 0.04 s	36 ± 12 s
C-Mantec Alg. execution time	0.06 ± 0.02 s	0.40 ± 0.32 s
Number representation	Integer	Floating point
Developing time	∼Months	∼Weeks

the creation of very compact neural architectures. The present work is an extension of Ref. [26] and is organized as follows: After a general introduction, Sect. 2 presents a comparative analysis of the advantages of FPGA technology for the implementation of neural networks. The manuscript then continues in Sect. 3 with a short description of the two neural network models analyzed, to later describe in Sect. 4 the hardware implementation details. Results from several comparison features are presented in Sect. 5, to finally present the discussion of the results and the conclusions obtained.

2 FPGAs as a Competitive Technology for the Implementation of Neural Networks

The advances in microelectronics in recent years have permitted the implementation of algorithms in different hardware platforms. One of these technologies, known as FPGAs, consists in a set of modifiable logic blocks with programmable interconnections. FPGAs, mainly programmed using a VHDL language have been widely used in telecommunications, and high technology applications like satellites, rovers, racing cars, etc. [3]. More recently, due to its increased capacity, price reduction and flexibility they have been also applied as robust hardware accelerator devices for the implementation of different kind of algorithms for vision processing, pattern recognition, signal processing, internet search, etc. [29]. Neuro-inspired algorithms, like neural networks, Deep Learning, Anfis networks, etc., are essentially distributed processing models where the computations are done in a parallel way, and so the full connectivity available in FPGAs for their logical components are an ideal device for their application. In Table 1 we indicate some specific features for the FPGA board used in this work (Model, price, energy consumption, execution times of the two algorithms used), together some general characteristics of FPGAs board like the standard number representation used and standard developing times. The table also shows the same parameters but for a standard PC.

A deeper analysis of the values shown in the table, mainly in relationship to the two algorithms compared is done later on this work, but from the information displayed in the table, in particular in relationship to execution times (expressed in seconds, s), it seems clear that FPGAs seem a very interesting technology for neural network approaches, as for a similar cost than a PC the execution times are on average 7 and 189 times shorter for C-Mantec and BP algorithms respectively, with the only disadvantage of the developing times that tend to be much longer for FPGA based implementations.

3 Theoretical Methods

3.1 The Backpropagation Algorithm

The backpropagation algorithm (BP) is a supervised learning method for training multilayer artificial neural networks. As the BP algorithm is very well known, we only give the essential details in order to understand the rest of the work.

Consider a neural network architecture comprising several hidden layers. If we consider the neurons belonging to a hidden or output layer, the activation of these units, denoted by y_i, can be written as:

$$y_i = g\left(\sum_{j=1}^{L} w_{ij} \cdot s_j\right) = g(h) \ , \tag{1}$$

where w_{ij} are the synaptic weights between neuron i in the current layer and the neurons of the previous layer with activation s_j. In the previous equation, h is the synaptic potential of a neuron, that includes a sigmoid type activation function.

The objective of the BP supervised learning algorithm is to minimize the difference between given outputs (targets) for a set of input data and the output of the network, measured by an error function (E) that depends on the values of the synaptic weights, and that can be defined as:

$$E = \frac{1}{2}\sum_{k=1}^{p}\sum_{i=1}^{M}(z_i(k) - y_i(k))^2, \tag{2}$$

where the first sum is on the p patterns of the data set and the second sum is on the M output neurons. $z_i(k)$ is the target value for output neuron i for pattern k, and $y_i(k)$ is the corresponding response output of the network. The synaptic weights between two last layers of neurons are updated as:

$$\Delta w_{ij}(k) = -\eta \frac{\partial E}{\partial w_{ij}(k)} = \eta[z_i(k) - y_i(k)]g_i'(h_i)s_j(k), \tag{3}$$

where η is the learning rate, g' is the derivative of the sigmoid function and h is the synaptic potential previously defined, while the rest of the weights are modified according to similar equations by the introduction of a set of values called the "deltas" (δ), that propagate the error from the last layer into the inner ones.

3.1.1 Training and Validation Processes

The training procedure is executed a certain number of times (epochs) using the training patterns. In one epoch, the training patterns are all presented once in random ordering, adjusting the synaptic weights in an on-line manner. A well-documented and severe problem affecting all predictive algorithms is the problem of overfitting, caused by an overspecialization of the training procedure on the training set of patterns [20]. In order to alleviate this effect, a straightforward strategy is to split the set of available training patterns in training, validation and test sets. The training set will then be used to adjust the synaptic weights according to Eq. 3, while the validation set is used to control overfitting effects, storing in memory the values of the synaptic weights that have so far led to the lowest validation error, so when the training procedure ends, the algorithm returns the stored set of weights. The test set is used to estimate the performance of the algorithm in unseen data patterns, as a way of estimating the generalization capability of the algorithm.

3.2 The C-Mantec Constructive Neural Network Algorithm

Competitive Majority Network Trained by Error Correction [34] (C-Mantec) is a novel neural network constructive algorithm that utilizes competition between neurons and a modified perceptron learning rule (thermal perceptron [9]) to build single hidden layer compact architectures with good prediction capabilities for supervised classification problems. The single hidden layer architectures constructed contains binary neurons in the hidden layer that are incorporated as the training process advances, and the output of the network consists of a single neuron that computes the majority function of the responses of the hidden neurons, like in a voting process.

The binary activation state (S_j) of the N_H neurons in the hidden layer depends on N input signals, ψ_i, and on the actual value of the N synaptic weights (ω_{ji}) and bias (b_j) as follows:

$$S = \begin{cases} 1\ (ON) & \text{if } h \geq 0 \\ 0\ (OFF) & \text{otherwise} \end{cases} \tag{4}$$

where h is the synaptic potential of the neuron defined as:

$$h = \sum_{i=1}^{N} \omega_{ji}\ \psi_i - b_j \tag{5}$$

In the thermal perceptron rule, the modification of the synaptic weights, $\Delta\omega_{ji}$, is done on-line (after the presentation of a single input pattern) according to the following equation:

$$\Delta\omega_{ji} = (t - S_j)\ \psi_i\ T_{fac}, \tag{6}$$

where t is the target value of the presented input (it is the same for all hidden neurons), and ψ_i represents the value of input unit i connected to the hidden neuron j by weight ω_{ji}. The difference to the standard perceptron learning rule is that the thermal perceptron incorporates the T_{fac} factor. This factor, whose value is computed as shown in Eq. 7, depends on the value of the synaptic potential and on an artificially introduced temperature (T).

$$T_{fac} = \frac{T}{T_0} e^{-\frac{h}{T}}, \tag{7}$$

The value of T decreases as the learning process advances according to Eq. 8, similarly to a simulated annealing process.

$$T = T_0 \cdot \left(1 - \frac{I}{I_{\max}}\right), \tag{8}$$

where I is a cycle counter that defines an iteration of the algorithm on one learning cycle, and $I_{\max}$ is the maximum number of iterations allowed. One learning cycle of the algorithm is the process that starts when a chosen pattern is presented to the network and finishes after checking that all neurons respond correctly to the input or when the synaptic weights of the neuron chosen to learn the actual pattern (whether an existing or a new neuron) modifies its synaptic weights.

The C-Mantec algorithm has three parameters to be set at the time of starting the learning procedure, and several experiments have shown the robustness of the algorithm that operates fairly well in a wide range of parameter values. The three parameters of the algorithm are:

– $I_{\max}$: maximum number of learning iterations allowed for each neuron in one learning cycle.

- g_{fac}: growing factor that determines when to stop a learning cycle and include a new neuron in the hidden layer.
- ϕ: determines in which case an input example is considered as noise and removed from the training dataset according to the following condition:

$$\text{delete}(x_i) \mid N_{LT} \geq (\mu + \phi \cdot \sigma), \tag{9}$$

where x_i represents an input pattern, N is the total number of patterns in the dataset, N_{LT} is the number of times that pattern x_i has been presented to the network on the current learning cycle, and where μ and σ corresponds to the mean and variance of the distribution for all patterns on the number of times that the algorithm has tried to learn each pattern in a learning cycle. The learning procedure starts with only one neuron included in the single hidden layer of the architecture and an output neuron that computes the majority function of the responses of the hidden neurons (a voting scheme). The process continues by presenting an input pattern to the network and if it is misclassified, it will be learned by one of the present neurons whose output did not match the target pattern value if certain conditions are met, otherwise a new neuron will be included in the architecture to learn it. Among all neurons that misclassified the input pattern, the one with the largest T_{fac} will learn it but only if this T_{fac} value is larger than the g_{fac} parameter, a condition included to prevent the unlearning of previous stored information. If no thermal perceptron meeting these criteria is found, a new neuron is added to the network, starting a new learning cycle that includes the resetting of the temperature of all neurons to T_0. The g_{fac} parameter has an important effect in the algorithm as directly controls the point at which new neurons are added to the hidden layer, affecting the final size of the obtained architectures. Also at the end of a cycle the noisy patterns filtering procedure (Eq. 9) is applied. This procedure consists in analyze the number of times an example has been presented to the network and required a synaptic weight correction (i.e., it was initially wrongly classified by the network), to eliminate at the end of a learning cycle those inputs that needed larger number of modifications in comparison to the mean. The algorithm continues its operation iteratively repeating the previous stages until all patterns in the training set are correctly classified by the network. A more detailed analysis of the C-Mantec algorithm can be found in Ref. [34].

4 Hardware Implementation

We describe in this section the main functioning aspects of both algorithms, including also specific details of the FPGA implementation. An important issue and a big difference in relationship to standard PC implementations regards the number representation used in the FPGA. While for standard PCs floating point number representation is the standard choice, this type is not usually the most efficient for FPGAs and a fixed point number representation is preferred [32]. The synaptic weights precision has been analyzed for the both algorithms (Back-Propagation and C-Mantec) in Sect. 4.4 in order to determinate the size and speed of the neural network model.

Regarding the FPGA implementation of the BP algorithm three main aspects have been carefully analyzed for increasing the efficiency of resource utilization: first, the introduction of a new input-hidden neurons block (Sect. 4.1), second, a new scheme for computing the sigmoid transfer function (Sect. 4.2), and third, a strategy of time division for using only a single multiplier block for each neuron (Sect. 4.3).

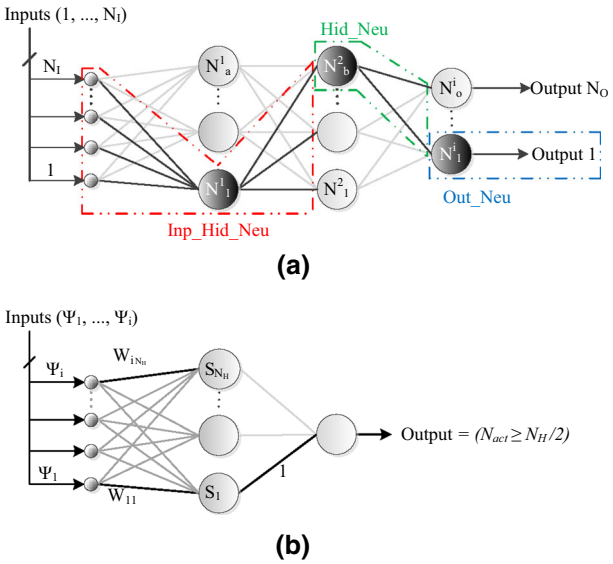

Fig. 1 Overall scheme of the FPGA architectures used for the implementation of the BP (**a**) and C-Mantec (**b**) algorithms

For the case of the implementation of the C-Mantec algorithm, the same aspects have been analyzed in order to carry out a fair comparison: the introduction of a novel architecture of a constructive neural network (Sect. 4.1), a scheme for implementing the majority function (Sect. 4.2), and also the same strategy of time division used in the BP case (Sect. 4.3).

4.1 FPGA Implementations of the Neural Network Architectures

Figure 1a, b show the two different schemes designed for the implementation of the neural architectures for the Back-Propagation (a) and C-Mantec (b) algorithms. The Back-Propagation architecture (Fig. 1a) is based on a new scheme where three types of blocks are used for the implementation of the different parts of the neural architecture. The proposed implementation does not consider the input layer of neurons separately as this is included together with the first hidden layer neurons in a module named input-hidden neurons ("inp-hid"). The definition of this new type of module is possible because the input layer neurons do not process the information as they simple act as input to the network, also it facilitates the calculation of the learning phase due to the neurons manage the synaptic weights of the next layer.

For the case of the C-Mantec algorithm (Fig. 1) the network architecture built contains a single hidden layer of threshold neurons (S_j) with output values $\{0, 1\}$. An important characteristic of this network is that permits a fast calculation of the output of the network since the information is not transmitted between hidden layers.

4.2 FPGA Implementation of Continuous Transfer Functions

Another important design aspect regarding the FPGA implementation of a neural network algorithm is the way of computing the activation function of the neurons, usually of a sigmoid-type, as in the case of the BP algorithm. A scheme based on a lookup table approach plus linear

interpolation scheme permits to obtain an efficient representation in terms of the resources needed together with low absolute and relative errors. In a previous work [27] a complete study about size of the lookup table, employed resources and precision results has been introduced, obtaining that efficient size values for the sigmoid function table consists in using 2 bits for the decimal part representation, 3 bits for the integer part and one extra bit for indicating the sign of the function. These parameters produce a table of $2^6 = 64$ inputs with 16 bits of word length for the the size of each input. The total resources necessary to implement this table are 32 bits or 1 block memory. Figure 2 shows the results obtained for the approximation of the sigmoid function (top graph) and the absolute and relative errors committed in its approximation (middle and bottom graphs).

For the case of the C-Mantec algorithm, the learning process involves the computation of the T_{fac} factor that requires the FPGA implementation of the exponential function. The same approach mentioned above for the computation of the sigmoid function was used. Section 5 includes Table 3 that shows a comparison between the approximation results obtained for both functions (sigmoid and exponential functions).

Further, the output of a C-Mantec network consists in a single output neuron (S) that computes the majority function (see Eq.10) of the activation of the hidden layer units, like in a voting process. The network output is active (1) if half or more of the N_H hidden neurons with activation values S_j are active (1):

$$S = \begin{cases} 1\ (ON) & \text{if } \sum S_j \geq \frac{N_H}{2} \\ 0\ (OFF) & \text{otherwise} \end{cases} \tag{10}$$

The FPGA implementation of the majority function is shown in Fig. 3. On the left part of the figure the activation value of all N_H hidden layer neurons S_i are shown, followed by the computation of the sum of their activations. In the module indicated as "comparator" the obtained value is compared with the value of $\frac{N_H}{2}$ and the whole network output is computed following Eq. 10. The whole process can be executed in less than one clock cycle of the FPGA because all operations involved are implemented with logic cells that introduce only minor delays.

4.3 Time Division Multiplexing Scheme for the Multiplier Block

Furthermore, a third important aspect considered during the FPGA implementation regards a time division scheme for performing the multiplications involved in the algorithm [27]. The multiplier blocks can be implemented both as a combination of logic cells or using specific DSP blocks. We have selected the first choice in this work for a fair comparison with the C-Mantec algorithm implementation, as this was done without DSPs. Further the implementation using DSP is specific to each board and thus the present choice gives more generality to the results. The strategy consists in using a single multiplier for each neuron (built using logic blocks [5]) and then through using a time division multiplexing scheme compute all the multiplications related to the neuron. The time division scheme for the multiplications can be observed in the Fig. 4, in which it is observed as a single multiplicator performs all multiplications outside of the logic of every neuron.

4.4 Synaptic Weights Precision

The representation of the synaptic weights can be chosen according to the available resources, taking into account that a higher accuracy requires a larger representation, which will imply

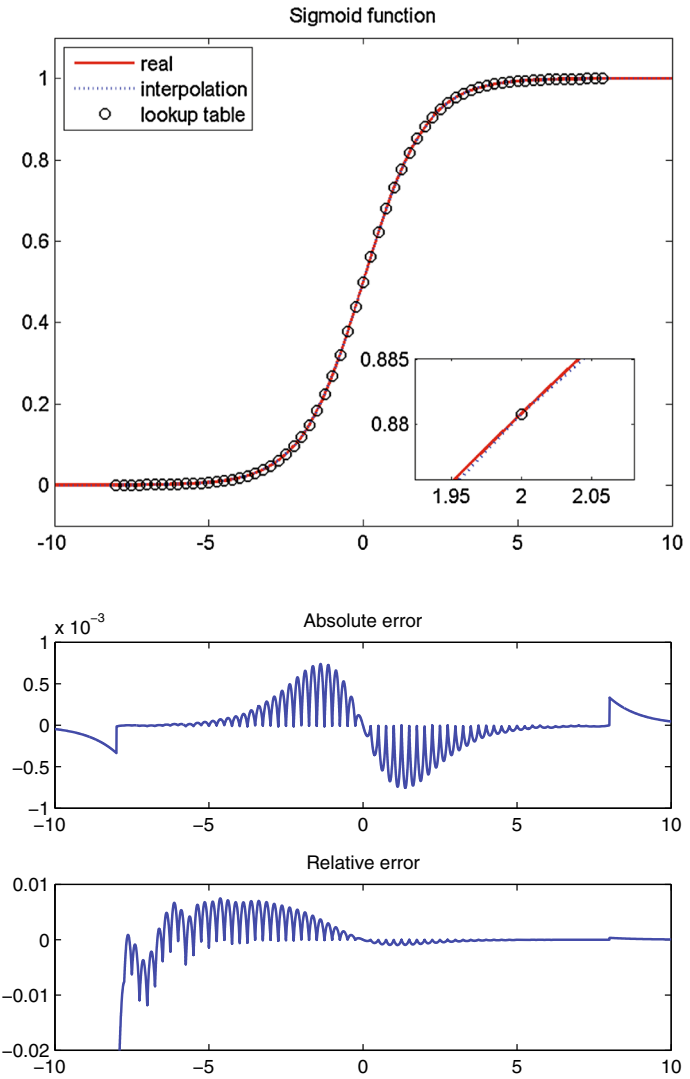

Fig. 2 Sigmoid function approximation results obtained using a lookup table plus linear interpolation scheme (*top graph*). Absolute (*middle graph*) and relative errors (*bottom graph*) committed in the approximation of the function

an increase in the number of LUTs per neuron (consequently a reduced number of available neurons) and a decrease in the maximum operation frequency of the FPGA board. Synaptic weight accuracy is important so that the resulting values are similar to those obtained with the floating point representation used in the PC based code. A synaptic weight is represented by a bit array with integer and fractional parts of length N_1 and N_2. N_1 determines the minimum and maximum values that can be obtained $-2^{(N_1-1)}$ to $2^{(N_1-1)}$ while N_2 defines the accuracy $2^{(-N_2)}$. The number of bits needed to represent all possible discrete values within a certain range of positive values depends on the difference between maximum and minimum values of the interval, and can be obtained from the following equation:

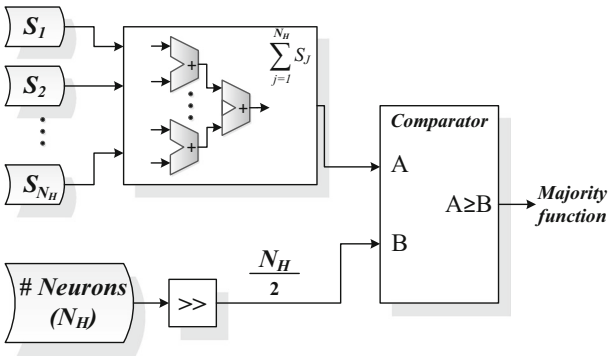

Fig. 3 Hardware implementation of the majority function needed for obtaining the output of a network trained by the C-Mantec algorithm

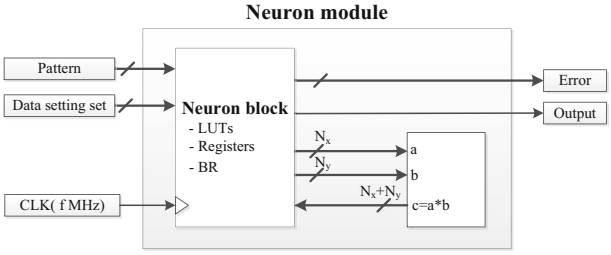

Fig. 4 Time division scheme for performing the multiplications involved in the algorithm, in which it is observed as a single multiplicator performs all multiplications outside of the logic of every neuron

$$\#\,\text{Bits} = \log_2((1 + \max(w_{ij}))/(\min(w_{ij}))). \tag{11}$$

Section 5 includes Table 3 that shows the number of LUTs per neuron, the number of available neurons and maximum operating for the both algorithms according to the number of bits used for representing the synaptic weights $N_1 + N_2$, where N_1 and N_2 indicate the integer and fractional parts of the representation.

5 Results

We present in this section results from the implementation of both algorithms (BP and C-Mantec) in a Virtex-5 OpenSPARC Evaluation Platform (ML509). This device includes a Xilinx Virtex-5 XC5VLX110T FPGA that provides different connector devices: 2 USB (Host and Peripheral) ports, 2 PS/2(Keyboard, Mouse) ports, RJ-45 (10/100/1000 Networking) and RS-232 (Male Serial port) connectors, 2 Audio Inputs (Line, Microphone), 2 Audio Outputs (Line, Amp, SPDIF), Video Input, Video Output (DVI/VGA), Single-Ended and Differential I/O Expansion. Table 2 shows some characteristics of the Virtex-5 XC5VLX110T FPGA, indicating its main logic resources. A picture of the Virtex-5 OpenSPARC Platform is shown in Fig. 5.

The VHSIC Hardware Description Language (VHDL) [2,5] language is used for programming the FPGA, under the "Xilinx ISE Design Suite 12.4" environment using the "ISim M.81d" simulator. VHDL is a hardware description language widely used in electronic design automation to describe digital and mixed-signal systems such as FPGAs and integrated circuits, and can be also used as a general purpose parallel programming language. Our design

Table 2 Main specifications of the Virtex-5 XC5VLX110T FPGA related to its available slice logic

Device	Slice registers	Slice LUTs	Bonded IOBs	Block RAM/FIFO
Virtex-5 XC5VLX110T	69,120	69,120	34	148

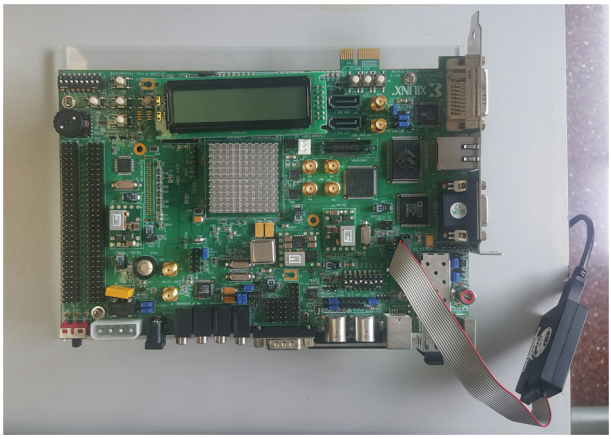

Fig. 5 Picture of the Virtex-5 XC5VLX110T board used for the implementation of the C-Mantec algorithm

Table 3 Maximum error (Max) and root mean square error (RMSE) for different values of N_a and N_b in the implementation of the exponential and sigmoidal functions

N_a	N_b	Exponential		Sigmoidal	
		Max	RMSE	Max	RMSE
3	3	1.833×10^{-3}	3.249×10^{-4}	3.353×10^{-4}	9.477×10^{-5}
3	4	4.704×10^{-4}	7.632×10^{-5}	1.982×10^{-4}	4.428×10^{-5}
4	4	4.704×10^{-4}	5.501×10^{-5}	6.091×10^{-5}	1.394×10^{-5}
4	5	1.151×10^{-4}	1.358×10^{-5}	2.669×10^{-5}	8.783×10^{-6}
4	6	2.530×10^{-5}	6.493×10^{-6}	1.755×10^{-5}	8.362×10^{-6}

strategy was to avoid the usage of specific Xilinx cores, in order to obtain a general design that can potentially be used in FPGAs from other manufacturers.

Table 2 shows some characteristics of the Virtex-5 XC5VLX110T FPGA, indicating its main logic resources. VHSIC Hardware Description Language (VHDL) [2,5] language was used for programming the FPGA, under the "Xilinx ISE Design Suite 12.4" environment using the "ISim M.81d" simulator.

Table 3 shows the Maximum and Root Mean Square errors for different values of the integer N_a and decimal parts N_b obtained for the implementation of the exponential and sigmoidal functions used in the C-Mantec and Back-Propagation algorithms respectively through a lookup table plus linear interpolation scheme. As it can be appreciated from the table both errors are quite low for both functions for almost all values of N_a and N_b being lower for the Sigmoidal function.

Table 4 Number of LUTs, maximum number of neurons and maximum frequency that can be implemented in a Virtex-5 board for the two algorithms and as a function of different fixed-point representations

N_1	N_2	LUTs/neuron		Max. # neurons		Frequency (MHz)	
		BP	C-M	BP	C-M	BP	C-M
8	8	787	689	82	94	73.9	74.1
8	12	967	757	67	85	53.0	74.1
8	16	1124	943	57	68	41.3	74.1
12	12	1057	826	61	78	52.7	53.7
12	16	1223	1033	53	62	41.1	53.7
16	16	1382	1299	47	50	40.9	42.7

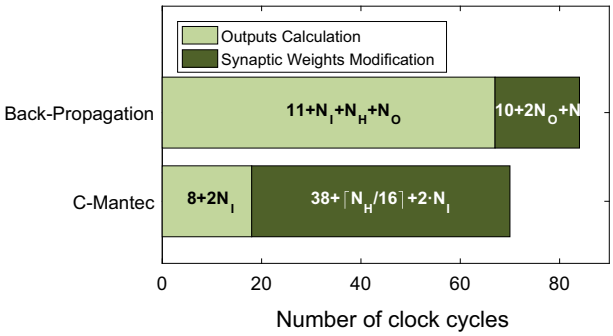

Fig. 6 Execution cycles needed to learn and compute the output for a single input pattern for the case of a 5-50-1 neural network architecture

One of the most interesting results of this work is the comparison carried out for obtaining the maximum number of neurons that can be implemented in the Virtex-5 board used, together with the analysis of the resources needed for the implementation of a single neuron for both algorithms. Table 4 shows the values obtained for BP and C-Mantec for different values for the integer N_1 and decimal part N_2 of the fixed point representations tested.

As a way of quantifying the time complexity of the implementation of both algorithms, we have also analysed the number of FPGA clock cycles involved in the computations related to training a network with one input pattern. The analysis done was divided in two parts: the number of cycles needed for the computation of the network output for a given input pattern, and the number of cycles related to the modification of the synaptic weights (cf. Eqs. 3, 6). The results displayed in Fig. 6 correspond to neural network architectures containing a single hidden layer with 50 neurons.

In order to get a more complete analysis, we have also calculated the computation times involved in the obtention of the output and in relationship to the learning process for a single pattern for both algorithms as the size of the neural network architecture increases. Figure 7 shows the computation times for this analysis as a function of the number of neurons in the hidden layer in which it can be observed that for the case of C-Mantec algorithms the computation times are almost constant, as a factor of 16 relates the computation time to the number of hidden neurons (see values in Fig. 6). The figure shows that for the BP algorithm the time grows linearly with the number of hidden neurons, and thus only for small architectures the Back-Propagation models operate faster than C-Mantec.

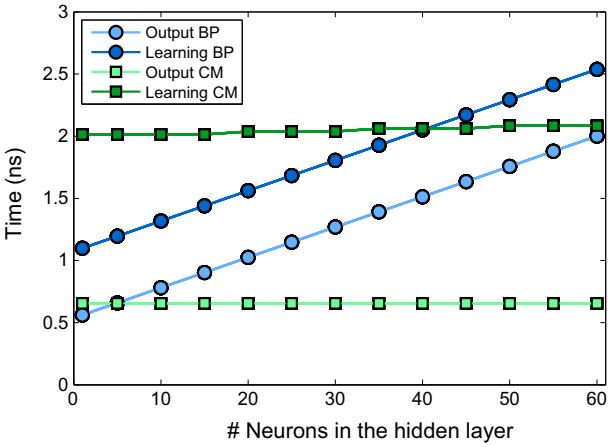

Fig. 7 Computation time for the calculation of the output and learning for a single input pattern depending on of the size of neural network architecture

Table 5 Generalization ability (%) obtained and number of neurons used in the architectures for the implementation of seven classic benchmark problems

Function	C-Mantec			Back-Propagation		
	Gen.	# Neu.	Time (ms)	Gen.	# Neu.	Time (ms)
Diabetes	76.6	5	97	79.3	5	227
Cancer	96.9	2	52	95.7	5	210
Heart	82.6	3	71	78.2	5	104
Ionosphere	87.4	2	56	87.5	5	210
Heart-c	82.5	2	55	80.1	5	190
Card	85.2	3	72	83.1	5	195
Sonar	75.0	1	43	75.2	5	223
Average	83.7	3	63	82.7	5	194

Using a set of benchmark functions from the UCI repository, we have computed the generalization ability and computation time (ms) for the execution of both algorithms. Table 5 shows these results together with the number of neurons used in each case, noting that C-Mantec sets this number automatically while a constant size architecture comprising 5 neurons was used for Back-Propagation.

6 Conclusions

We have presented and analyzed the implementation in a FPGA board of two neural network learning algorithms: the traditional Back-Propagation model and C-Mantec a constructive neural network model. The algorithms operate from different principles as BP is essentially a gradient based algorithm minimizing an error function for a fixed architecture that has to be defined in advance, while C-Mantec is an error correcting method that constructs the network architecture automatically as it learns the input patterns. In terms of the FPGA implementa-

tion, both methods require the implementation of continuous functions (the sigmoid and the exponential functions), process that is very simple for standard computers (PCs) but much more complex for hardware devices using a fixed point number representation like FPGAs. An analysis of the resources needed to implement both functions efficiently indicate that similar error levels are obtained for both cases when using a lookup table plus linear interpolation scheme, with slightly lower error values for the case of the sigmoid function used in the BP algorithm. Nevertheless, it is worth noting that as C-Mantec is an error correcting algorithm the precision needed for the arithmetic representation is lower than for BP, that requires higher precision levels as its algorithm involves an accurate computation of the derivatives of the neural activation functions for its correct operation.

Another very important issue regarding the comparison of both algorithms is the amount of hardware resources needed for the implementation of single neurons in both algorithms, and in this aspect the advantage is on the C-Mantec side as a lower number of LUTs is required, permitting for a given board the construction of larger neural network architectures, For the case analyzed in this work (Virtex-5 XC5VLX110T board) the results averaged over different number representation (cf. Table 4) implies a 18.7% increase in the maximum number of neurons that can be included in the architectures for C-Mantec in comparison to the BP algorithm. Further, we have also estimated the average computation time needed for training both algorithms using a set of benchmark functions, finding that C-Mantec operates faster than BP, needing in average a third of the computational time (cf. Table 5). A more detailed analysis on the number of cycles and the time needed for the computation of processes involved in obtaining the output neuron value and in the modification of the synaptic weights, cf. Figs. 6 and 7 shows a different behavior for both algorithms, with a clear advantage favoring C-Mantec for the case of larger architectures.

As a conclusion, the present work shows a detailed comparison regarding the possibilities of the application of two neurocomputational algorithms using FPGA boards. From general functioning consideration and from the results obtained for both algorithms in terms of integer number representation, hardware resources and computation times, C-Mantec seems more suited for its FPGA implementation as first it does not need the a priori specification of the neural architecture to use, and second as it is less demanding in terms of hardware resource utilization and computation times.

Acknowledgements The authors acknowledge support from Junta de Andalucía (Secretaría General de Universidades, Investigación y Tecnología) through Grant P10-TIC-5770, and from MINECO (Spain) through Grants TIN2010-16556 and TIN2014-58516-c2-1-R (all including FEDER funds).

References

1. Akin E, Aydin I, Karakose M (2011) FPGA based intelligent condition monitoring of induction motors: detection, diagnosis, and prognosis. In: Proceedings IEEE International Conference on Industrial Technology (ICIT 2011), pp 373–378. doi:10.1109/ICIT.2011.5754405
2. Ashenden P (2008) The designer's guide to VHDL, vol 3, 3rd edn. Morgan Kaufmann Publishers Inc., San Francisco (Systems on Silicon)
3. Bacon D, Rabbah R, Shukla S (2013) FPGA programming for the masses. Queue 11:40–52
4. Chappell GJ, Lee J, Taylor JG (1992) A review of medical diagnostic applications of neural networks. Springer, London
5. Chu PP (2008) FPGA prototyping by VHDL Examples: Xilinx Spartan-3 version. Wiley, New York
6. Conmy P, Bate I (2010) Component-based safety analysis of FPGAs. IEEE Trans Industr Inform 6(2):195–205

7. Cui H, Zhang H, Ganger GR, Gibbons PB, Xing EP (2016) Geeps: scalable deep learning on distributed gpus with a gpu-specialized parameter server. In: ACM European conference on computer systems (EuroSys'16)
8. Franco L, Elizondo D, Jerez J (2009) Constructive neural networks. Springer, Berlin
9. Frean M (1990) The upstart algorithm: a method for constructing and training feedforward neural networks. Neural Comput 2(2):198–209
10. Gómez I, Franco L, Jerez JM (2009) Neural network architecture selection: Can function complexity help? Neural Process Lett 30(2):71–87
11. Gomperts A, Ukil A, Zurfluh F (2011) Development and implementation of parameterized FPGA-based general purpose neural networks for online applications. IEEE Trans Industr Inform 7(1):78–89
12. Gu R, Shen F, Huang Y (2013) A parallel computing platform for training large scale neural networks. In: 2013 IEEE international conference on big data, pp 376–384
13. Guresen E, Kayakutlu G, Daim TU (2011) Using artificial neural network models in stock market index prediction. Expert Syst Appl 38(8):10,389–10,397
14. Haykin S (1994) Neural networks: a comprehensive foundation. Prentice Hall, Englewood Cliffs
15. Karayiannis NB, Venetsanopoulos AN (1993) Applications of neural networks: a review. Springer, Boston
16. Kilts S (2007) Advanced FPGA design: architecture, implementation, and optimization. Wiley-IEEE Press, New York
17. Le Q, Jeon J (2010) Neural-network-based low-speed-damping controller for stepper motor with an FPGA. IEEE Trans Industr Appl 57:3167–3180
18. Mehrotra K, Mohan CK, Ranka S (1997) Elements of artificial neural networks. MIT Press, Cambridge
19. Meireles MRG, Almeida PEM, Simoes MG (2003) A comprehensive review for industrial applicability of artificial neural networks. IEEE Trans Industr Electron 50(3):585–601
20. Hawkins M (2003) A comprehensive. IEEE Trans Industr Electron 50(3):585–601
21. Monmasson E, Idkhajine L, Cirstea M, Bahri I, Tisan A, Naouar MW (2011) FPGAs in industrial control applications. IEEE Trans Industr Inform 7(2):224–243
22. Najafabadi MM, Villanustre F, Khoshgoftaar TM, Seliya N, Wald R, Muharemagic E (2015) Deep learning applications and challenges in big data analytics. J Big Data 2(1):1–21
23. Omondi A, Rajapakse J (2006) FPGA implementations of neural networks. Springer, New York
24. Orlowska-Kowalska T, Kaminski M (2011) FPGA implementation of the multilayer neural network for the speed estimation of the two-mass drive system. IEEE Trans Industr Inform 7(3):436–445
25. Ortega-Zamorano F, Jerez J, Franco L (2014) FPGA implementation of the C-Mantec neural network constructive algorithm. IEEE Trans Industr Inform 10(2):1154–1161
26. Ortega-Zamorano F, Jerez J, Juarez G, Franco L (2015) FPGA implementation comparison between C-Mantec and Back-Propagation neural network algorithms. In: Lecture notes in computer science, vol 9095
27. Ortega-Zamorano F, Jerez J, Juarez G, Perez J, Franco L (2014) High precision FPGA implementation of neural network activation functions. In: 2014 IEEE symposium on intelligent embedded systems (IES), pp 55–60
28. Ortega-Zamorano F, Jerez JM, Urda-Munoz D, Luque-Baena RM, Franco L (2016) Efficient implementation of the backpropagation algorithm in FPGAs and microcontrollers. IEEE Trans Neural Netw Learn Syst 27:1840–1850
29. Putnam A, Caulfield AM, Chung ES, Chiou D, Constantinides K, Demme J, Esmaeilzadeh H, Fowers J, Gopal GP, Gray J, Haselman M, Hauck S, Heil S, Hormati A, Kim JY, Lanka S, Larus J, Peterson E, Pope S, Smith A, Thong J, Xiao PY, Burger D (2014) A reconfigurable fabric for accelerating large-scale datacenter services. In: Proceeding of the 41st annual international symposium on computer architecuture, ISCA '14. IEEE Press, Piscataway, pp 13–24. http://dl.acm.org/citation.cfm?id=2665671.2665678
30. Reed RD, Marks RJ (1998) Neural smithing: supervised learning in feedforward artificial neural networks. MIT Press, Cambridge
31. Rumelhart D, Hinton G, Williams R (1986) Learning representations by back-propagating errors. Nature 323(6088):533–536
32. Savich A, Moussa M, Areibi S (2007) The impact of arithmetic representation on implementing MLP-BP on FPGAs: a study. IEEE Trans Neural Netw 18(1):240–252
33. Serbedzija NB (1996) Simulating artificial neural networks on parallel architectures. Computer 29(3):56–63
34. Subirats JL, Franco L, Jerez JM (2012) C-Mantec: a novel constructive neural network algorithm incorporating competition between neurons. Neural Netw 26:130–140
35. Tiwari V, Khare N (2015) Hardware implementation of neural network with sigmoidal activation functions using cordic. Microprocess Microsyst 39(6):373–381. doi:10.1016/j.micpro.2015.05.012

36. Valtierra-Rodriguez M, Osornio-Rios R, Garcia-Perez A, Romero-Troncoso R (2013) FPGA-based neural network harmonic estimation for continuous monitoring of the power line in industrial applications. Electr Power Syst Res 98:51–57. doi:10.1016/j.epsr.2013.01.011
37. Werbos PJ (1974) Beyond regression: new tools for prediction and analysis in the behavioral sciences. PhD thesis, Harvard University
38. Yasunaga M, Yoshida E (1998) Optimization of parallel BP implementation: training speed of 1056 mcups on the massive. In: The 1998 IEEE international joint conference on neural networks proceedings, 1998. IEEE world congress on computational intelligence, vol 1, pp 563–568
39. Zhu J, Sutton P (2003) FPGA implementations of neural networks—a survey of a decade of progress. Lect Notes Comput Sci 2778:1062–1066