FPGA Implementation Comparison Between C-Mantec and Back-Propagation Neural Network Algorithms

Francisco Ortega-Zamorano[1], José M. Jerez[1], Gustavo Juárez[2], and Leonardo Franco[1](✉)

[1] Department of Computer Science, ETSI Informática, Universidad de Málaga, Málaga, Spain
{fortega,jja,lfranco}@lcc.uma.es
[2] Universidad Nacional de Tucumán, San Miguel de Tucumán, Argentina
gjuarez@herrera.unt.edu.ar

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 Back-Propagation algorithm and C-Mantec, a constructive neural network algorithm that generates compact one hidden layer architectures. 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 analysed, focusing in features like logic and memory resources needed, transfer function implementation, computation time, etc. Advantages and disadvantages of both methods are discussed in the context of their application to benchmark problems.

Keywords: Constructive neural networks · FPGA · Hardware implementation

1 Introduction

Artificial Neural Networks (ANN) [1] 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, etc. The implementation of ANN in digital circuits (standard PCs, embedded systems, etc.) has been limited by the computational power needed, mainly given its intrinsic parallelism. In this sense, the capacity and performance of current FPGAs are a realistic alternative for the real time implementation of ANN. FPGAs [2] 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. Recent advances in technology have permitted to construct FPGAs with considerable large amounts of processing power and memory storage, and as so they have been applied in several domains (Telecommunications, Robotics, Pattern recognition tasks, Infrastructure monitoring, etc.) [3–5]. In particular FPGAs seem quite suitable for Neural Network implementations as they can be programmed to operate in a parallel way [6–8].

Within the area of supervised pattern recognition, the efficient implementation of neurocomputational models into hardware can be done in principle following two different strategies: Modifying and adapting the traditional Back-Propagation algorithm or developing new algorithms that are 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 [9], and secondly through the implementation of an alternative algorithm (C-Mantec) based on an incremental constructive architecture [10]. The overall idea of the work is to do a comparative analysis of the two approaches in order to identify the pros and cons for each case, and with this information take a decision depending on the specific application and the resources available. The Back-Propagation algorithm (BP) is the standard learning procedure for training multilayer neural networks architectures [11] [12] 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. On the other hand, C-Mantec [13] is a novel neural network constructive algorithm that utilizes competition between the neurons and a modified perceptron learning rule (thermal perceptron [14]) to build single hidden layer compact architectures with good prediction capabilities for the supervised classification problems. The organization of the present work is as follows: Section 2 includes the hardware implementation details and description about the Back-propagation and C-Mantec algorithms. Results from several comparison features are presented in Section 3, to finally present the discussion of the results and the conclusions obtained.

2 Algorithms Description and Implementation Details

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 [15].

2.1 The Back-Propagation Algorithm

The Back-Propagation algorithm is a supervised learning method for training multilayer artificial neural networks based on the gradient descent strategy.

The network architecture for the implementation of the algorithm has to be decided previously and there is no standard methodology for this step, being the trial-and-error method one of the most used strategies. In the most general case the neural architecture comprises an input layer with a number of inputs determined by the problem at hand, several hidden layers, and one or many output neurons depending whether a binary or multi-output problem is analysed (for simplicity the first case is considered in this work).

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. This error depends on the values of the synaptic weights, and so these should be adjusted in order to minimize the error. The error function computed for the case of a single output neuron can be defined as:

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

where the sum is over all training patterns, and z_i and y_i refers to target and network outputs for a given pattern i.

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:

$$y_i = g\left(\sum_{j=1}^{L} w_{ij} \cdot s_j\right) = g(h)\ , \tag{2}$$

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, we have introduced h as the synaptic potential of a neuron. g is a sigmoid activation function given by:

$$g(x) = \frac{1}{1 - e^{-\beta x}} \tag{3}$$

By using the method of *gradient descent*, the BP algorithm attempts to minimize the error Eq. 1 in an iterative process by updating the synaptic weights upon the presentation of a given pattern. The synaptic weights between two last layers of neurons are updated as:

$$\Delta w_{ij}(k) = -\eta\frac{\partial E}{\partial w_{ij}(k)} = \eta[z_i(k) - y_i(k)]g'_i(h_i)s_j(k), \tag{4}$$

where η is the learning rate that has to be set in advance (a parameter of the algorithm), 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 form the last layer into the inner ones.

The BP neural networks have been trained under a training / validation / testing strategy to avoid overfitting effects, caused mainly by excessive training

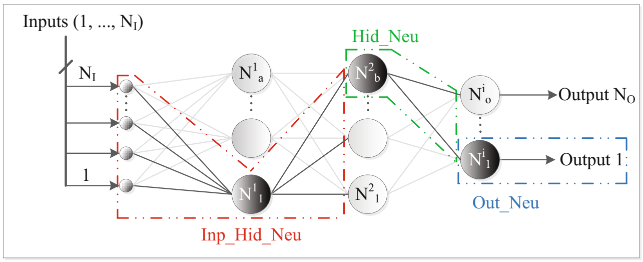

Fig. 1. Scheme of the FPGA architecture design for the implementation of the BP algorithm

iterations in the learning process, and thus a validation set is used to check the evolution of the mean square error between target and output values.

Regarding the FPGA implementation of the BP algorithm three main aspects have been carefully analyzed for increasing the efficiency of resource utilization: a) The introduction of a new input-hidden neurons block, b) A new scheme for computing the sigmoid transfer function, and c) A strategy of time division for using only a single multiplier block for each neuron.

Fig. 1 shows a scheme of the neural architecture, where three types of blocks are used for the implementation of the different parts of the neural architecture. The proposed implementation do 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.

Another important FPGA design aspect regarding the hardware implementation of a neural network algorithm is the way of computing the activation function of the neurons, usually a sigmoid-type function. An 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 the previous work [16] a complete study about size of the table, employed resources and precision results has been presented. As conclusion the more efficient dimension of the table for a sigmoid is 2 bits for the decimal part, 3 bits for the integer part and one more bit foe the sign of the function. This parameters produce a table of $2^6 = 64$ inputs with 16 bits of word length as size of each input. The total resources necessary to implement this table are 32 bits or 1 block memory. Plus, Fig. 2 shows the approximation obtained for the sigmoid function (top graph) and the errors committed in its approximation (bottom graph).

Furthermore, a third important aspect considered during the FPGA implementation regards a time division scheme for performing the multiplications involved in the algorithm [16]. 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

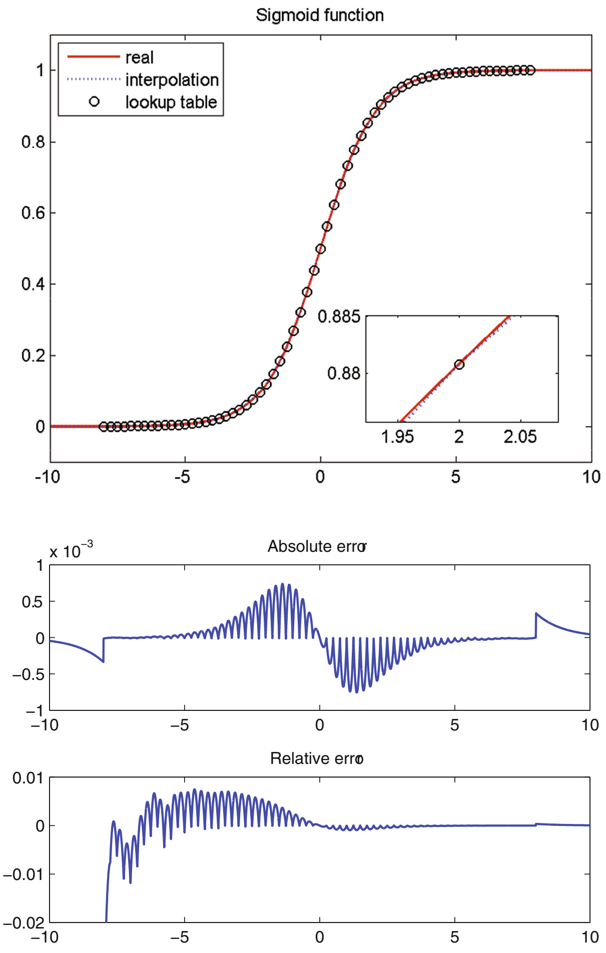

Fig. 2. FPGA sigmoid function approximation based on a lookup table plus linear interpolation scheme (top graph). Absolute (middle graph) and relative errors (bottom graph) committed in the approximation of the function.

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 [17]) and then through using a time division multiplexing scheme compute all the multiplications related to the neuron.

2.2 The Constructive Neural Network Algorithm C-Mantec

C-Mantec as a constructive neural network algorithm generates the network topology in an on-line manner during the learning phase, avoiding the complex problem of selecting an adequate neural architecture [13]. 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 the creation of very compact neural architectures. The binary activation state (S_j) of each of the 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:

$$S_j = \begin{cases} 1 & \text{if } h_j \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

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

$$h_j = \sum_{i=1}^{N} \omega_{ji}\ \psi_i - b_j \tag{6}$$

The weight updating in the C-Mantec algorithm at the single neuron level is done using the thermal perceptron rule [14], in which the modification of the synaptic weights, $\Delta\omega_i$, is done on-line (after the presentation of a single input pattern) according to the following equation:

$$\Delta\omega_{ji} = (t - S_j)\ \psi_i\ T_{fac}, \tag{7}$$

where t is the target value (desired output of the whole network for the presented input), and ψ represents the value of input unit i connected to the hidden neuron S_j by synaptic 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. 8, depends on the value of the synaptic potential and on an artificially introduced temperature (T):

$$T_{fac} = \frac{T}{T_0} e^{-\frac{|h|}{T}}, \tag{8}$$

The computation of the T_{fac} factor involves the FPGA implementation of the exponential function, task that was done using the same approach applied for the computation of the sigmoid function needed for the BP algorithm. Section 3 includes Table 2 that shows a comparison between the approximation results obtained for both functions (the sigmoid and exponential functions).

Following with the description of the C-Mantec algorithm, the value of the temperature T decreases as the learning process advances according to Eq. 9, similarly to a simulated annealing process.

$$T = T_0 \cdot (1 - \frac{I}{I_{max}}), \tag{9}$$

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

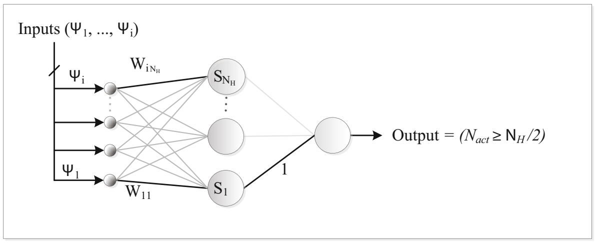

Fig. 3. Example of network architecture constructed by the C-Mantec algorithm

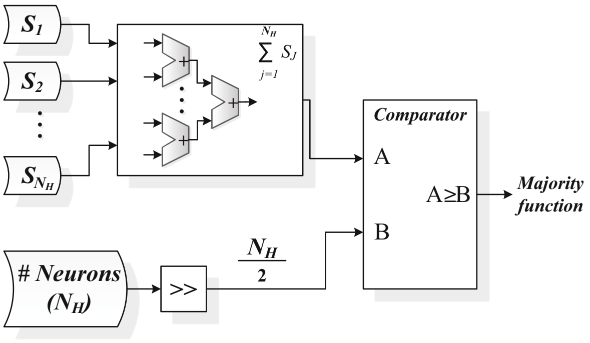

Fig. 4. Hardware implementation of the majority function that corresponds to the output neuron activation function of a network trained by the C-Mantec algorithm

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

$$\text{Output} = \begin{cases} 1 & \text{if } \sum_j^{N_H} S_j \geq \frac{N_H}{2} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

Fig. 3 shows a network architecture of the type built by the C-Mantec algorithm. The network contains a single hidden layer of threshold neurons (S_j) with output values $\{0, 1\}$.

The FPGA implementation of the majority function is shown in Fig. 4. 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 activation. In the module indicated by "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.

3 Results

We present in this section results from the implementation of both algorithms (BP and C-Mantec) in a Xilinx Virtex-5 board. Table 1 shows some characteristics of the Virtex-5 XC5VLX110T FPGA, indicating its main logic resources. VHDL [17,18] (VHSIC Hardware Description Language) language was used for programming the FPGA, under the "Xilinx ISE Design Suite 12.4" environment using the "ISim M.81d" simulator.

Table 1. Main specifications of the Xilinx Virtex-5 XC5VLX110T FPGA board

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

Table 2 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 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 2. 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	$1.833 \cdot 10^{-3}$	$3.249 \cdot 10^{-4}$	$3.353 \cdot 10^{-4}$	$9.477 \cdot 10^{-5}$
3	4	$4.704 \cdot 10^{-4}$	$7.632 \cdot 10^{-5}$	$1.982 \cdot 10^{-4}$	$4.428 \cdot 10^{-5}$
4	4	$4.704 \cdot 10^{-4}$	$5.501 \cdot 10^{-5}$	$6.091 \cdot 10^{-5}$	$1.394 \cdot 10^{-5}$
4	5	$1.151 \cdot 10^{-4}$	$1.358 \cdot 10^{-5}$	$2.669 \cdot 10^{-5}$	$8.783 \cdot 10^{-6}$
4	6	$2.530 \cdot 10^{-5}$	$6.493 \cdot 10^{-6}$	$1.755 \cdot 10^{-5}$	$8.362 \cdot 10^{-6}$

One of the most interesting results of this work is the comparison done for the maximum number of neurons that can be implemented with the board used, and also the resources associated with the implementation of a single neuron in

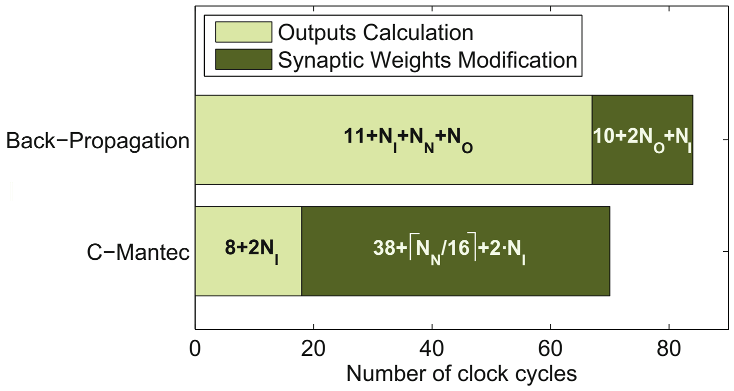

Fig. 5. Execution cycles needed to learn and compute the output for a single input pattern for the case of a 5-50-1 neural network architecture

relationship to both algorithms. Table 3 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.

Table 3. Number of LUTs and maximum number of neurons 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		# Neurons	
		BP	C-M	BP	C-M
8	8	787	689	82	94
8	12	967	757	67	85
8	16	1124	943	57	68
12	12	1057	826	61	78
12	16	1223	1033	53	62
16	16	1382	1299	47	50

We have also analysed the number of FPGA clock cycles involved in the computations related to training a network with one input pattern. The total number of cycles is divided in two parts related to the number cycles related to compute the output of the network for a given input and for the modification of the synaptic weights (cf. Eqs. 4 and 7). The results displayed in the table correspond to two neural networks with 50 neurons in the single hidden layer of the architecture.

Using a set of benchmark functions from the UCI repository, we have computed the generalization ability and computation time (ms) for both algorithms. Table 4 shows these results together with the number of neurons used in each

Table 4. 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

case, noting that C-Mantec sets this number automatically while a constant size architecture comprising 5 neurons was used for Back-Propagation.

4 Conclusion

We have presented and analysed the implementation in a FPGA board of two neural network learning algorithms: Back-Propagation and C-Mantec. The algorithms operates from different principles as BP is a gradient based algorithm minimizing an error function for pre-determined 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 implementation, both methods require the implementation of continues 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 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 that for BP that might require high precision levels as it involves the accurate computation of the derivatives of the 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, that in our case resulted in approximately a 18.7% increase in the maximum number of neurons that can be included (cf. Table 3). 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 4).

As an overall conclusion the present work shows a comparison regarding the possibilities of the application of neurocomputational algorithms using FPGA boards. The comparison of both algorithms is a little bit in favour of C-Mantec as first it does not need the a priori specification of the neural architecture to be used, and second as it is less demanding in terms of hardware resource utilization.

Acknowledgments. The authors acknowledge support from Junta de Andalucía through grants P10-TIC-5770 and P08-TIC-04026, and from CICYT (Spain) through grant TIN2010-16556 (all including FEDER funds).

References

1. Haykin, S.: Neural networks: a comprehensive foundation. Prentice Hall (1994)
2. Kilts, S.: Advanced FPGA Design: Architecture, Implementation, and Optimization. Wiley-IEEE Press (2007)
3. Monmasson, E., Idkhajine, L., Cirstea, M., Bahri, I., Tisan, A., Naouar, M.W.: Fpgas in industrial control applications. IEEE Transactions on Industrial Informatics **7**(2), 224–243 (2011)
4. Bacon, D., Rabbah, R., Shukla, S.: Fpga programming for the masses. Queue **11**, 40–52 (2013)
5. Conmy, P., Bate, I.: Component-based safety analysis of fpgas. IEEE Transactions on Industrial Informatics **6**(2), 195–205 (2010)
6. Zhu, J., Sutton, P.: Fpga implementations of neural networks - a survey of a decade of progress. In: Cheung, P.Y.K., Constantinides, G.A. (eds.) FPL 2003. LNCS, vol. 2778, pp. 1062–1066. Springer, Heidelberg (2003)
7. Gomperts, A., Ukil, A., Zurfluh, F.: Development and implementation of parameterized fpga-based general purpose neural networks for online applications. IEEE Trans. Industrial Informatics **7**(1), 78–89 (2011)
8. Le, Q., Jeon, J.: Neural-network-based low-speed-damping controller for stepper motor with an fpga. IEEE Transactions on Industrial Applications **57**, 3167–3180 (2010)
9. Ortega-Zamorano, F., Jerez, J., Urda, D., Luque-Baena, R., Franco., L.: Efficient implementation of the backpropagation algorithm in fpgas and microcontrollers. IEEE Transactions on Neural Networks and Learning Systems (2015) (in press)
10. Ortega-Zamorano, F., Jerez, J., Franco, L.: Fpga implementation of the c-mantec neural network constructive algorithm. IEEE Transactions on Industrial Informatics **10**(2), 1154–1161 (2014)
11. Werbos, P.J.: Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. PhD thesis, Harvard University (1974)
12. Rumelhart, D., Hinton, G., Williams, R.: Learning representations by back-propagating errors. Nature **323**(6088), 533–536 (1986)
13. Subirats, J.L., Franco, L., Jerez, J.M.: C-mantec: A novel constructive neural network algorithm incorporating competition between neurons. Neural Netw. **26**, 130–140 (2012)
14. Frean, M.: The upstart algorithm: a method for constructing and training feedforward neural networks. Neural Computation **2**(2), 198–209 (1990)

15. Savich, A., Moussa, M., Areibi, S.: The impact of arithmetic representation on implementing mlp-bp on fpgas: A study. IEEE Transactions on Neural Networks **18**(1), 240–252 (2007)
16. Ortega-Zamorano, F., Jerez, J., Juarez, G., Perez, J., Franco, L.: High precision fpga implementation of neural network activation functions. In: 2014 IEEE Symposium on Intelligent Embedded Systems (IES), pp. 55–60, December 2014
17. Chu, P.P.: FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Version. John Wiley & Sons (2008)
18. Ashenden, P.: The Designer's Guide to VHDL (Systems on Silicon), vol. 3, 3rd edn. Morgan Kaufmann Publishers Inc., San Francisco (2008)