Engineering Applications of Artificial Intelligence 30 (2014) 179–188

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Smart sensor/actuator node reprogramming in changing environments using a neural network model

Francisco Ortega-Zamorano [a,*], José M. Jerez [a], José L. Subirats [a], Ignacio Molina [b], Leonardo Franco [a]

[a] Department of Computer Science, E.T.S.I. Informatica, University of Malaga, Bulevar Louis Pasteur, 35, 29071 Malaga, Spain
[b] Max Planck Institute, Munich, Germany

ARTICLE INFO

Article history:
Received 17 September 2013
Received in revised form
23 December 2013
Accepted 9 January 2014
Available online 1 February 2014

Keywords:
Constructive Neural Networks
Microcontroller
Arduino

ABSTRACT

The techniques currently developed for updating software in sensor nodes located in changing environments require usually the use of reprogramming procedures, which clearly increments the costs in terms of time and energy consumption. This work presents an alternative to the traditional reprogramming approach based on an on-chip learning scheme in order to adapt the node behaviour to the environment conditions. The proposed learning scheme is based on C-Mantec, a novel constructive neural network algorithm especially suitable for microcontroller implementations as it generates very compact size architectures. The Arduino UNO board was selected to implement this learning algorithm as it is a popular, economic and efficient open source single-board microcontroller. C-Mantec has been successfully implemented in a microcontroller board by adapting it in order to overcome the limitations imposed by the limited resources of memory and computing speed of the hardware device. Also, this work brings an in-depth analysis of the solutions adopted to overcome hardware resource limitations in the learning algorithm implementation (e.g., data type), together with an efficiency assessment of this approach when the algorithm is tested on a set of circuit design benchmark functions. Finally, the utility, efficiency and versatility of the system is tested in three different-nature case studies in which the environmental conditions change its behaviour over time.

1. Introduction

Sensors (or detectors) are devices that permit the measurement of chemical or physical variables, transforming them into electrical signals, in order to interpret the signals from various sensors and send the sensed data or make a decision according to them. Microcontroller boards are an economic, small and flexible solution, and thus are the most common controller used in a sensor node, also known as a Mote, commonly used in Wireless sensor network (Yick et al., 2008; Sengupta et al., 2013), but also used in other important technologies such as Embedded systems (Marwedel, 2006; Mamdoohi et al., 2012) and Real-time systems (Kopetz, 1997; Wang et al., 2010). Motes are nowadays widely employed in all kind of industrial applications, in several of them, the problem requires an action upon the environmental conditions, in this case a sensor/actuator node is required.

In cases when the environmental conditions evolve over time, the original sensor/actuator programming can lead to incorrect decisions, and thus it is necessary to change or adapt the decision-making process to the new conditions (Sayed-Mouchaweh and Lughofer, 2012). The traditional option to solve this problem has been to send the sensed data to a central unit, where a person interprets the data and reprogram the microcontroller with the new set of rules (Han et al., 2005; Wang et al., 2006; Shaikh et al., 2010). Different reprogramming techniques have been proposed as a way of dynamically changing the behaviour of the sensors without having to manually reprogram them, because traditional reprogramming requires in most cases the interruption of the process for loading the new binary code, with the consequent loss of time and energy, involved in the communication process to the central unit (Rassam et al., 2013; Aiello et al., 2011). A first step towards reducing the previous effects has been to incorporate machine learning systems in the decision-making process, automating the response of the microcontroller without interrupting its execution and sending just a small fraction of code to the microcontroller (Urda et al., 2012; Canete et al., 2012; Farooq et al., 2010). However, recent advances in the computing power of sensors permit the inclusion of learning systems in the microcontroller ("on-chip" learning), adapting the sensor/actuator behaviour dynamically according to the sensed data (Aleksendrić et al., 2012; Mahmoud et al., 2013).

Artificial Neural Networks (ANNs) (Haykin, 1994) are a kind of machine learning models, inspired on the functioning of the brain,

* Corresponding author. Tel.: +34 952 13 28 47; fax: +34 952 131 397.
E-mail addresses: fortega@lcc.uma.es, FOZamorano@gmail.com (F. Ortega-Zamorano).

0952-1976/$ - see front matter © 2014 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.engappai.2014.01.006

that can be utilised in clustering and classification problems, having been applied successfully in several fields, including pattern recognition (Dhanalakshmi et al., 2011), stock market prediction (Park and Shin, 2013), control tasks (Zhai and Yu, 2009), medical diagnosis and prognosis (Kodogiannis et al., 2007), and so on. Despite years of research in the field of ANN, selecting a proper architecture for a given problem remains a difficult task (Gómez et al., 2009; Hunter et al., 2012; Lakshmi and Subadra, 2013), and several strategies have been proposed for solving or alleviating this issue. In particular, Constructive Neural Networks (CoNNs) offer the possibility of generating networks that grows as input information is received, matching the complexity of the data (Franco et al., 2009). Moreover, the training procedure in CoNN, considered a computationally expensive problem in standard feedforward neural networks, can be done on-line and relatively fast. C-Mantec is a recently introduced CoNN algorithm that implements competition between neurons, also incorporating a built-in filtering scheme to avoid overfitting problems. These two characteristics permit the algorithm to generate compact neural architectures with very good generalisation capabilities, making the algorithm suitable for its application to devices with limited resources like microcontrollers. The main limitations of these devices are memory size and computing speed, and thus an efficient implementation of the algorithm is needed. Despite the existence of alternative evolving models (Lughofer, 2011; Angelov, 2010; Huang et al., 2005), C-Mantec has been selected based mainly on the three following features: dynamic generation of compact architectures, good prediction ability and robustness to parameter setting.

In the present work, we have fully implemented the C-Mantec (Subirats et al., 2012) constructive neural network model in an Arduino UNO board, including the whole learning process to implement the automatic reprogramming process for decision-making into the sensor/actuator in changing environments, avoiding communication to other devices.

The Arduino UNO board was used (Oxer and Blemings, 2009) as it is a popular, economic and efficient open source single-board microcontroller that allows easy project development (Lian et al., 2013; Cela et al., 2013; Ortega-Zamorano et al., 2013; Kornuta et al., 2013). We have also proposed three case studies of different nature which require reprogramming to the decision-making process, demonstrating that the time involved in the sensor reprogramming is significantly lower than in the traditional case, without the need to send any information to another device (as a control unit), saving a large fraction of the required energy resources.

The paper is structured as follows: first, we briefly describe in Section 2 the C-Mantec constructive neural network algorithm used, followed by a description of the Arduino UNO microcontroller board in Section 3. Section 4 includes the details of the implementation with the results obtained shown in Section 5. Thereafter, three case studies are evaluated in Section 6, checking the efficiency of the system in them, to finally extract the conclusions in Section 7.

2. C-Mantec, constructive neural network algorithm

C-Mantec (Subirats et al., 2012) (Competitive Majority Network Trained by Error Correction) is a novel neural network constructive algorithm that utilises competition between neurons and a modified perceptron learning rule (thermal perceptron Frean, 1990) to build single hidden layer compact architectures with good prediction capabilities for supervised classification problems. As a CoNN algorithm, C-Mantec generates the network topology on-line during the learning phase, avoiding the complex problem of selecting an adequate neural architecture. 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) 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_i) and bias (b) as follows:

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

where h is the synaptic potential of the neuron defined as

$$h = \sum_{i=0}^{N} \omega_i \psi_i \quad (2)$$

In the thermal perceptron rule, 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:

$$\Delta\omega_i = (t - S)\psi_i T_{fac}, \quad (3)$$

where t is the target value of the presented input, and ψ represents the value of input unit i connected to the output by weight ω_i. The difference in 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. (4), depends on the value of the synaptic potential and on an artificially introduced temperature (T).

$$T_{fac} = \frac{T}{T_0} e^{-|h|/T}, \quad (4)$$

The value of T decreases as the learning process advances according to Eq. (5), similar to a simulated annealing process.

$$T = T_0 \cdot \left(1 - \frac{I}{I_{max}}\right), \quad (5)$$

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 random chosen pattern is presented to the network and finishes after checking that the output of the network is equal to the target for this pattern, or when a chosen neuron (the neuron with largest T_{fac} value or a new added neuron) modifies its synaptic weights to learn the actual presented pattern.

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 algorithm has the following three parameters:

- 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 includes 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:

$$\text{delete}(x_i) | N_{LT} \geq (\mu + \phi\sigma), \quad (6)$$

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 correspond 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 one neuron present 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 of the algorithm, a condition included to prevent the unlearning of previous stored information. If no thermal perceptron meeting these criteria are found, a new neuron is added to the network, starting a new learning cycle that includes the resetting of all neurons temperature to T_0. Also at the end of a cycle the noisy patterns filtering procedure (Eq. (6)) is applied. The algorithm continues its operation iteratively repeating the previous stages until all patterns in the training set are correctly classified by the network. During the learning process catastrophic forgetting is prevented as synaptic weights are only modified if the change involved is small (controlled by the value of g_{fac} and by an annealing process that reduces the temperature as learning proceeds), as if this is not the case the algorithm introduces a new neuron in the architecture (Subirats et al., 2012).

The Flowchart of C-Mantec algorithm is shown in Fig. 1, the most relevant function is represented in boxes, the decision-making in diamonds and the most important states (start and finish) in ovals.

3. The Arduino UNO board

Arduino is a single-board microcontroller designed to make the process of using electronics in multidisciplinary projects more accessible. The hardware consists of a simple open source hardware board designed around an 8-bit Atmel AVR microcontroller, though a new model has been designed around a 32-bit Atmel ARM. The software consists of a standard programming language compiler and a boot loader that executes on the microcontroller.

Arduino is a descendant of the open-source *Wiring* platform and is programmed using a Wiring-based language (syntax and libraries); similar to C++ with some slight simplifications and modifications, and a processing-based integrated development environment. Arduino boards can be purchased pre-assembled or do-it-yourself kits, and hardware design information is available. The maximum length and width of the Arduino UNO board are 6.8 and 5.3 cm respectively, with the USB connector and power jack extending beyond the former dimension.

The Arduino UNO is based on the ATmega328 chip (Atmel, Datasheet 328). It has 14 digital input/output pins, which can be used as input or outputs, and in addition, has some pins for specialised functions, for example 6 digital pins can be used as PWM outputs. It also has 6 analogy inputs, each of which provide 10 bits of resolution, together with a 16 MHz ceramic resonator, USB connection with serial communication, a power jack, an ICSP header, and a reset button. The ATmega328 chip has 32 KB of memory storage (0.5 KB are used for the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM. A picture of the Arduino UNO board is shown in Fig. 2.

The Arduino UNO has a communication protocol for its interaction with a computer, with another Arduino board or other microcontrollers. The ATmega328 provides serial communication (UART) which is available on digital pins 0 (RX) and 1 (Tx), also has I2C and SPI communication.

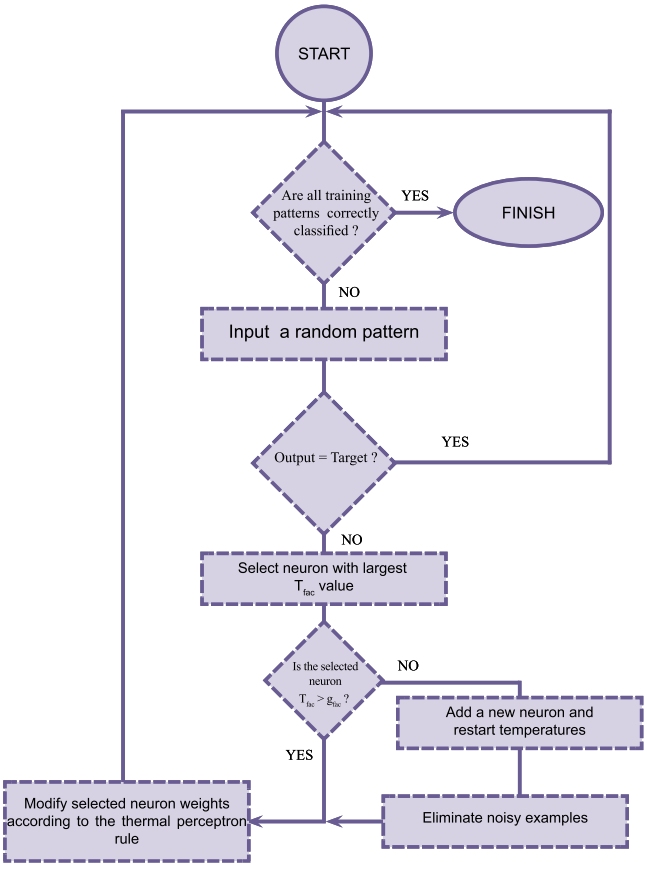

Fig. 1. Flow diagram of the C-Mantec constructive learning algorithm.

4. Implementation of the C-Mantec algorithm

The neural network model comprises two phases (learning and execution). In the learning phase, the synaptic weights for the neural network are computed from a set of patterns stored in the memory of the microcontroller, while in the execution phase, the microcontroller obtains the response to sensed input data according to the previously learned model. The learning phase comprises two different states (loading of input patterns and neural network learning). Data can be loaded into the microcontroller EEPROM memory on-line by I/O pins or by a serial communication USB port, but in both cases, the patterns have to be stored into the EEPROM memory, after, the neural network learning state starts.

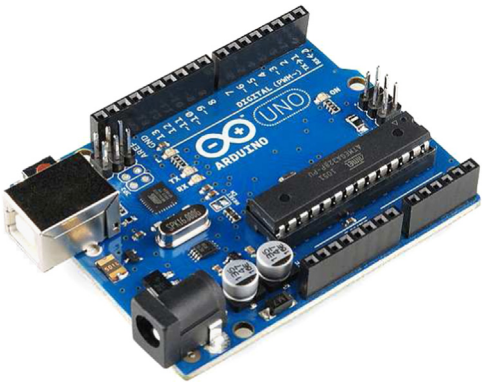

Fig. 2. Picture of an Arduino UNO board used for the implementation of the C-Mantec algorithm.

We explain next, the main technical issues considered for the implementation of the algorithm according to the two learning modes mentioned before:

4.1. Loading of patterns

The patterns have to be stored in the memory microcontroller because the learning process works in cycles and uses the pattern set repeatedly. For Boolean functions, it is only necessary to store the true value function outputs because the inputs are represented by the memory position. For example, for the pattern "01101001"→ '0', the input "01101001" corresponds to the decimal number 105 and thus the value '0' is stored in the position of memory 105. The microcontroller has 1 KB of EPROM memory, i.e., 8192 bits (2^{13}) limiting the number of Boolean inputs to 13. For the case of using an incomplete truth table, because the noisy pattern elimination stage is used or because of the nature of the function, the memory is divided into two parts, a first one that corresponds to the function outputs and a second part to indicate the inclusion or not of a given pattern in the learning set. In the case of using an incomplete truth table the maximum size of the input dimension is reduced to 12.

For the case of using real-value patterns is necessary to know in advance the actual number of bits that are used to represent each input variable. Eight bits have been used to represent each variable, taking into account that these values have to be normalised between 0 and 255. The following equation permits to compute the maximum number of input patterns:

$$N_P \cdot N_I + N_P/8 \leq 1024, \tag{7}$$

where N_I is the number of inputs and N_P is the number of patterns. N_P depends on the number of entries and the number of bits used for each entry.

4.2. Neural network learning

C-Mantec is an algorithm which adds neurons when they are become necessary, action that is not easily implemented in microcontroller, because the use of memory is static so the maximum number of neurons, which are stored in SRAM, must be previously defined. From this memory, with a capacity of 2 KB, we will employ less than 1 KB for storing the variables of the program; and thus saving at least 1 KB of free memory for saving the variables related to the neurons.

The microcontrollers are devices with limited computing speed so for obtaining more velocity in the learning process we have changed the data type of the variables associated to neurons. The floating point representation is the usual data type used in this kind of algorithm to represent all variables but this representation is not the most efficiency. We have selected fixed point to represent the variables associated to neurons for this we have to change the data type of these variables to integer. This paradigm shift has incited to essential changes in the way to program this algorithm but in return a higher learning speed and a smaller size of each variable has been obtained.

Regarding the variables associated to neurons, it is given next some details about the representation used:

- T_{fac}: Represented as a *float* type and occupying 4 bytes.
- Number of iterations: An integer value with a range between 1000 and 65,535 iterations, so a 2 bytes *integer* variable was used.
- Synaptic weights: Almost all calculations are based on these variables, so to speed up the computations an *integer* variable of 2 bytes long was used.
- Synaptic potential (h): It is calculated as a result of a summation of the synaptic weights, so not to saturate this value a type *long* of 4 bytes in length is used.

According to the previous definitions, the maximum number of neurons (N_N) that can be implemented should verify the following constraint:

$$4 \cdot N_N + 2 \cdot N_N + 2 \cdot N_N \cdot (N_I + 1) + 4 \cdot N_N \leq 1024, \tag{8}$$

where N_I is the number of inputs. For a worst case (maximum number of inputs is 13), the maximum number of neurons is 28. If during the learning process, the architecture reaches this maximum number of neurons, the algorithm finishes and the sensor outputs an error message.

The synaptic weights and synaptic potential have been implemented with 10 bits precision for the decimal part, so that the value of the weights will be between 32 and −32. Integer data types with values between −32,768 and 32,767 were used. The representation of the synaptic potential is done in a similar way, except that for this value 4 bits were used, allowing values between 2,097,152 and −2,097,152. The computation of T_{fac} is done using a float data type because it involves an exponential operation that can only be done with this type of data, but as its computation involves integer values, two different conversions must be done. The first conversion is done in Eq. (3), where T_{fac} values must be converted to fixed point representation, operation done by multiplying the T_{fac} value by 1024. The second conversion is performed in Eq. (4) where the synaptic potential (h) has to be converted to floating point representation for the calculation of an exponential function. In this case, ten right logical shifts were used in the synaptic potential, to then convert its data type to floating point for the final calculation of the T_{fac}. In order to avoid the saturation of the synaptic weights value, when any of these values are larger than 32 or lower than −32, all weights are divided by two (a right logical shift) when any of them reach an absolute value of 30. This change does not affect the algorithm execution because neural networks are invariant to this kind of scaling.

5. Results

We first tested the correct functioning of the microcontroller implementation of the C-Mantec algorithm comparing to the original published results (Subirats et al., 2012) in terms of the number of neurons generated in the neural network architectures, analysing as well the execution time for the microcontroller using floating point and integer representation. A set of 14 Boolean functions have been used for the comparison, including 12 single output functions from the MCNC circuits testing benchmark plus two XOR functions with two and three inputs. The C-Mantec algorithm was run with the following parameter configuration: g_{fac}=0.05 and I_{max}=1000. The results are shown in Table 1 where the first two columns indicate the function reference name and its number of inputs, and the third, fourth and fifth columns show that the number of neurons obtained in the original C-Mantec publication (Subirats et al., 2012), and the integer and floating point microcontroller implementation, respectively. Two last columns in Table 1 show the learning times in seconds (s) for the integer and floating point implementation of the microcontroller. The displayed values (mean ± standard deviation) are computed from 50 random samples.

Fig. 3 shows the learning time for all 14 benchmark functions included in Table 1 both for the integer and floating point implementation. The top graph in the figure shows on a logarithmic scale for the y-axis the learning time while the bottom graph shows the comparative speed between both representations used (integer and floating point).

Further, Fig. 4 shows the temporal evolution for the most complex analysed function (Alu2k), where the top graph shows the execution times related to the addition of a new neuron in the

Table 1
Number of neurons and learning time of the synthesis of a set of functions for fixed-point (integer) and floating point implementations compared with the original paper.

Funct.	# Inp.	# Neurons			Time (s)	
		Theory	Integer	Floating	Integer	Floating
XOR2	2	2.0 ± 0.0	2.0 ± 0.0	2.0 ± 0.0	0.36 ± 0.01	0.57 ± 0.03
XOR3	3	3.0 ± 0.0	3.0 ± 0.0	3.0 ± 0.0	1.4 ± 0.11	2.22 ± 0.26
cm82af	5	3.0 ± 0.0	3.0 ± 0.0	3.0 ± 0.0	1.84 ± 0.28	3.15 ± 0.64
cm82ag	5	3.0 ± 0.0	3.6 ± 0.6	3.6 ± 0.7	4.11 ± 1.88	8.10 ± 4.41
cm82ah	5	3.0 ± 0.0	3.0 ± 0.0	3.0 ± 0.0	1.85 ± 0.21	3.35 ± 0.63
z4ml24	7	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.23 ± 0.05	0.47 ± 0.13
z4ml25	7	3.1 ± 0.7	3.1 ± 0.4	3.1 ± 0.4	3.36 ± 1.04	6.43 ± 2.05
z4ml26	7	3.0 ± 0.0	3.0 ± 0.0	3.0 ± 0.0	2.67 ± 0.45	5.39 ± 1.11
9symml	9	3.0 ± 0.0	3.0 ± 0.0	3.0 ± 0.0	4.43 ± 0.58	12.8 ± 2.6
alu2k	10	11.2 ± 0.9	12.7 ± 0.9	12.3 ± 0.7	220 ± 50	969 ± 260
alu2m	10	2.0 ± 0.0	2.0 ± 0.0	2.0 ± 0.0	3.94 ± 0.21	13.1 ± 0.9
alu2n	10	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.41 ± 0.15	0.99 ± 0.41
alu2o	10	11.2 ± 0.9	13.0 ± 0.8	12.4 ± 0.8	312 ± 59	1444 ± 824
alu2p	10	3.0 ± 0.0	3.0 ± 0.0	3.0 ± 0.0	20.8 ± 43.7	84.1 ± 17.2

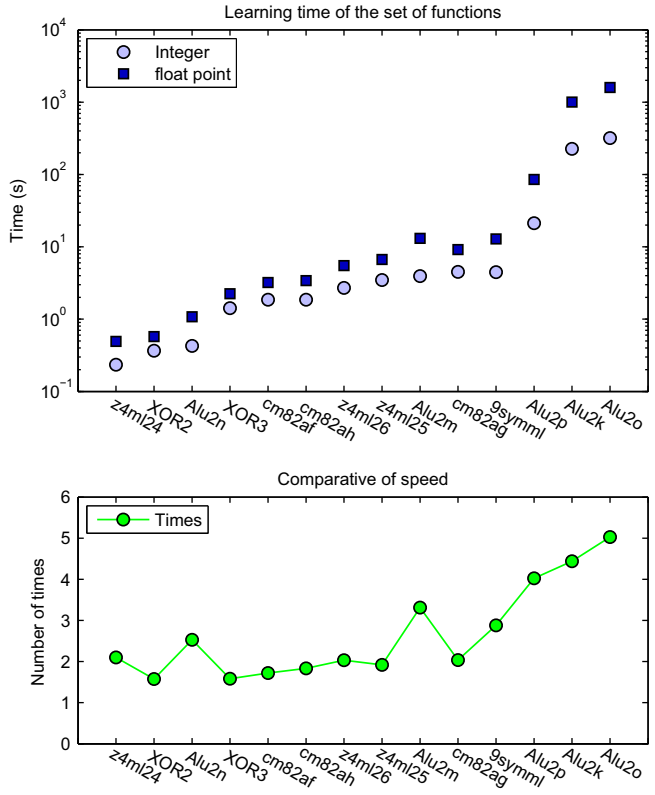

Fig. 3. Learning time (in seconds, logarithmic scale) for each function of the Boolean function benchmark data set with the two data type representation (top graph) and the relative speed between them (bottom graph).

constructive network architecture while Fig. 4 bottom shows the relative speed between the two implementations (integer and floating point). The time shown in the top graph of Fig. 4 includes several learning cycles until an input pattern wrongly classified by the network is presented and there is no neuron in the architecture that can be selected to learn it (according to the values of T_{fac} and g_{fac}).

The C-Mantec algorithm includes three procedures that are modified depending on the data type (floating or integer) representation used. These three functions that can be observed in Fig. 1 are the following: *Input a random pattern*, *Select neuron with largest* T_{fac} *value*, and *Modify selected neuron weights*. The algorithm

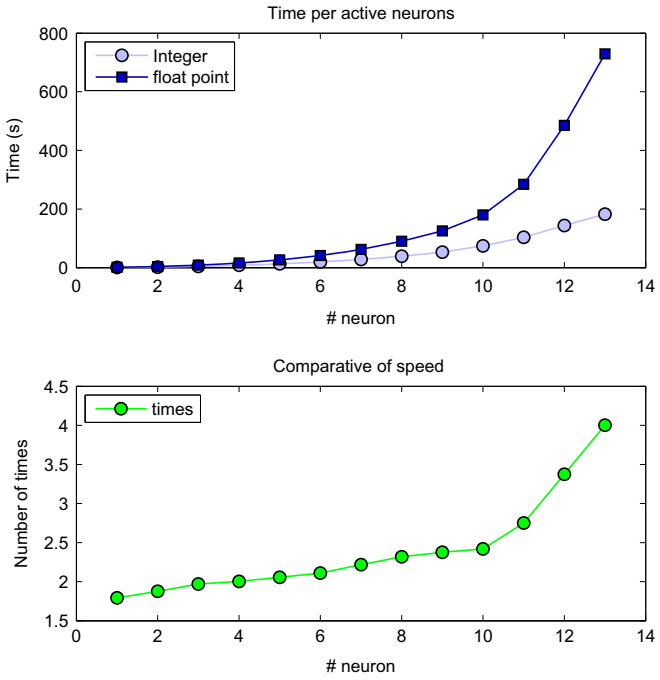

Fig. 4. Temporal evolution for the function Alu2k related to the addition of a new neuron to the network architecture. Absolute time (top graph) and relative time (bottom graph) for both representations (integer and floating point) used.

includes other two procedures (*Add a new neuron* and *Eliminate noisy examples*) that are not altered if the data representation is changed.

For each of the procedures that change according to the representation used we computed the mean execution time averaged across 50 samples for both cases. Fig. 5 shows the results for each procedure for integer and floating point representations. Fig. 5a shows the whole network execution time as the number of neurons increases from 1 to 20 for different input pattern dimensions (2, 5, 10, 15), while Fig. 5b shows a comparison between both implementations, where it can be appreciated the number of times that the integer representation is faster than the floating point one. Fig. 5c and d shows the execution time and relative comparison (respectively) for the computation of the maximum value of the T_{fac} for both representations as the number of neurons present in the architecture increases from 1 to 20. Fig. 5e and f shows the same as the previously described case but for the procedure *Modify selected neuron weights*.

6. Case studies

In this section, in order to test the efficiency of using neural networks in applications where microcontrollers for sense and/or act on the environment might be required, three case studies of different nature have been considered. The first case deals with detecting fires by an alarm system that can be reprogrammed according to the security level required. The second analysed case is a control system for the activation of a solenoid valve that depends on environmental variables, while the third case corresponds to a person fall detection problem with the constraint of using only a limited set of data patterns. The parameter settings of the C-Mantec algorithm were the same for all analysed cases: $g_{fac}=0.05$, $I_{max}=1000$ and $\phi=2$. We verified that changes on these values did not improve significantly the performance on each of the problems and then decided to use this set of parameter in all three cases. We note that it has been previously verified that

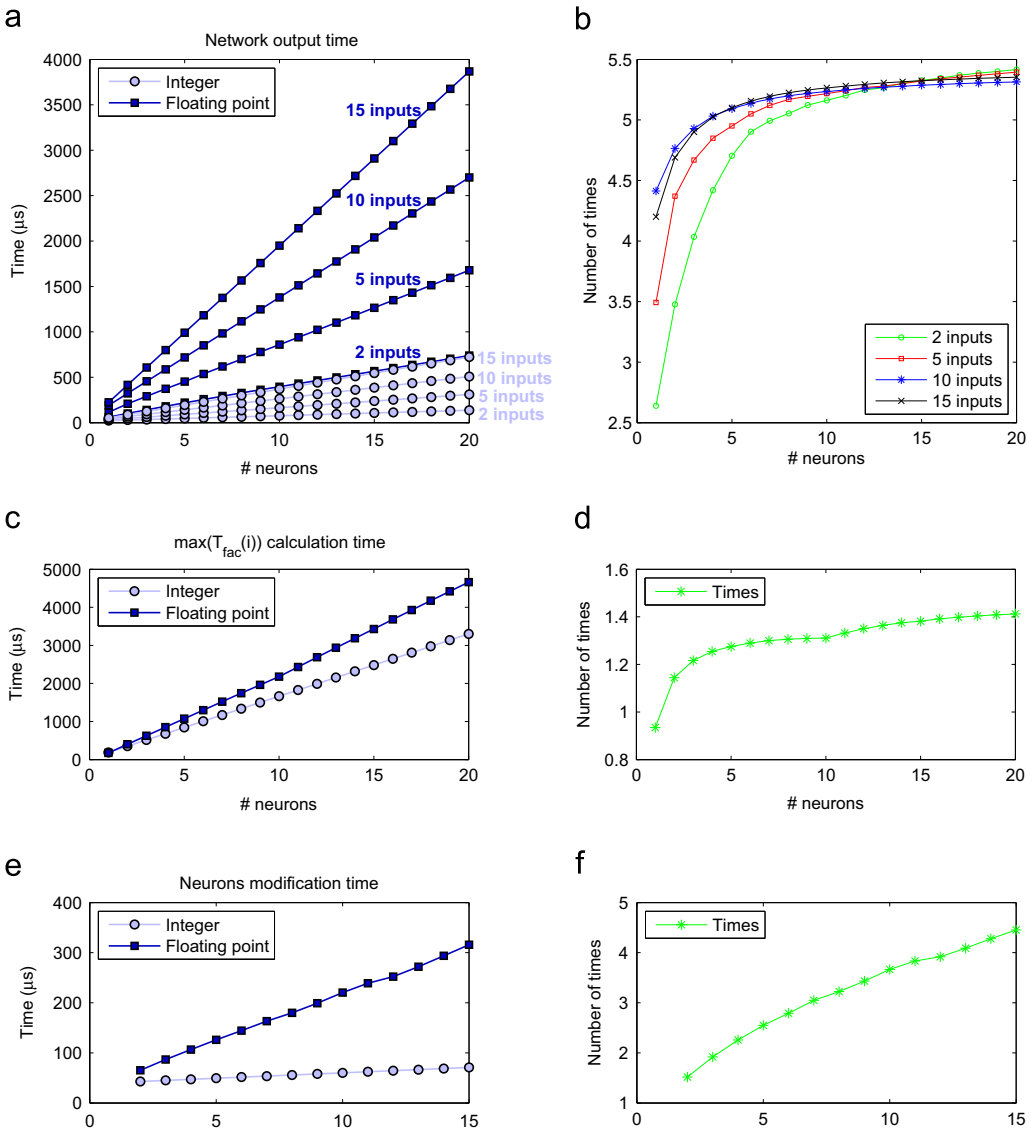

Fig. 5. Execution times of the three procedures included in the C-Mantec algorithm that changed depending on the data type representation used. Graphs in the first column show the absolute times measured in μs while the graphs in the second column show a comparison between both execution times (see the text for more details).

C-Mantec is very robust against changes on the parameter values (Subirats et al., 2012, 2013; Urda et al., 2013; Luque-Baena et al., 2013).

6.1. Fire alarm system

Fire alarm systems are widely used in inhabited premises. We analyse the case of a system whose security levels can be reprogrammed according to the user convenience, by defining activation thresholds as a function of the sensed environmental variables (temperature, smoke and gas). A schematic drawing of a room where a fire alarm is installed is shown in Fig. 6 including the three typical variables that can affect the system.

We have represented the different states of the fire alarm system using a truth table which is shown in Table 2.

Each different sensor (Gas, Smoke and Temperature) can be active ('1') or not ('0'), and these levels determine the Alarm state in the range from 1 to 8. As C-Mantec is a binary classifier, the alarm levels were codified in binary notation, as indicated in the last column of the table. A given functioning state of the alarm system includes setting the alarm levels for each of the detectors states, i.e., choosing the alarm level for a given sensor inputs.

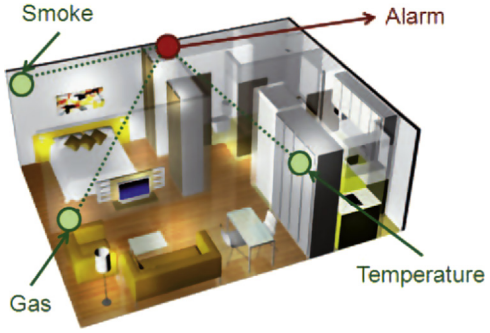

Fig. 6. Schematic representation of a room in which is installed a fire alarm.

Table 2 corresponds to a case in which each alarm level is different for every different input, as this is the worst possible case.

We first checked the time employed by the system to learn the initial state, measuring also the number of neurons included by the C-Mantec constructive algorithm and the energy consumption. Table 3 shows the results averaged across 50 samples, where

Table 2
Truth table of the initial state of the system for fire detection.

Gas	Smoke	Temperature	Alarm	Representation
0	0	0	1	0 0 0
0	0	1	2	0 0 1
0	1	0	3	0 1 0
0	1	1	4	0 1 1
1	0	0	5	1 0 0
1	0	1	6	1 0 1
1	1	0	7	1 1 0
1	1	1	8	1 1 1

Table 3
Time, energy consumption and number of neurons in the constructed neural network architecture for learning the initial state of a reprogrammable alarm system.

# Neurons	Time (ms)	Consumption (nAh)
3.0 ± 0.0	8.2 ± 4.8	73.1 ± 43.4

execution time is measured in milliseconds (ms) while energy consumption is measured in nano Amperes per hour (nAh).

Next, we have tested the time needed to relearn a modification of the state of the alarm, that corresponds to a change of values in the fourth column in Table 2. The results are shown in Table 4 where mean, maximum and minimum reprogramming time are displayed together with the measured consumption.

6.2. Weather prediction

Weather prediction is a relevant issue in human daily life, related for example for making good decisions in agriculture (moment of harvest, time of watering and crop type). Weather prediction is a very complex problem and neural network based system has been widely used for this task (Taylor and Buizza, 2002; Chakraborty et al., 2004).

We have consider a system that consists in a sensor/actuator that measures five environmental variables (Temperature (T), Wind speed (W_s), Wind direction (W_d), Humidity (H) and Solar Irradiance (I)) and takes a decision, which can be to irrigate or not the surrounding land. Fig. 7 displays a real picture of a constructed sensor/actuator node that may be used for the described problem.

To analyse a possible scenario where this node may be used, we consider the case of determine whether to water or not the surrounding land according to the value of the five environmental variables mentioned before. For the implementation these variables have been discretised using a 12 bits representation where the number of bits used for each variable can be seen from Table 5 together with the discretisation intervals.

The evaluation of the implementation of the neural network model for controlling the actuator system has been carried out starting from a null initial state. Afterwards, random input conditions were considered. An example of a generic instance can be:

$$\text{if}(T = 17.5\ ^\circ\text{C}) \wedge (W_s = 2\ \text{m/s}) \wedge (W_d = \text{“N”}) \wedge (H = 55\%) \wedge (I = 900\ \text{W/m}^2) \rightarrow \text{solenoid valve “open”}.$$

In the previous case, the indicated instance corresponds to a input of the truth table of the form “0100 00 00 10 11” with output ‘1’ as it was indicated that the solenoid valve, controlling the watering system, should open for the indicated condition. The whole truth table for the current discretisation used comprises 4096 different instances.

Table 4
Mean, maximum and minimum time involved in reprogramming a microcontroller and its corresponding energy consumption for a fire alarm system.

	Max	Min	Mean
Time (ms)	20.27	3.13	10.97
Consumption (nAh)	180.17	27.82	97.51

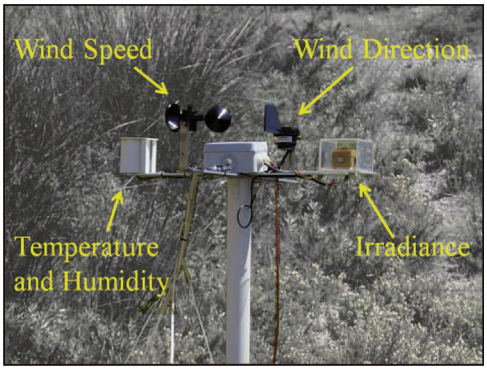

Fig. 7. Picture of a sensor/actuator system responsible for sensing different environmental variables and acting accordingly.

Table 5
Environmental variables used in a weather prediction problem, indicating the number of bits used in their representations and the discretisation intervals used.

Var.	# bits	Discretisation
T	4	[−2.5, 0, 2.5, 5, 7.5, 10, 12.5, 15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35] °C
W_s	2	[0, 2, 5, 9] m/s
W_d	2	N: [335°–45°), E: [45°–135°), S: [135°–225°), W: [225°–335°)
H	2	[5, 30, 55, 80]%
I	2	[100, 300, 600, 900] W/m^2

We measured the evolution of the time needed by the system as the number of defined instances is increased, together with the level of accuracy. Fig. 8 top shows time and consumption averaged across 30 random executions (Mean) and also one randomly chosen execution of the system in order to appreciate the existing variability of the learning process. Accuracy, computed as the fraction of correct responses over the whole truth table is shown in table Fig. 8 bottom. Note that to compute the accuracy, we suppose that the final whole truth table is known at every given time, even if the actuator only gets this information one pattern at a time, implying that in practice the accuracy (as it is computed) can be only analysed at the end of the process. If we had computed the accuracy over the presented pattern, this would have always been equal to 1.

A singular behaviour in the initial instances can be observed in Fig. 8, the neural network model has to be modified because almost all instances are misclassified thus at the beginning of the process, the learning time is higher and the accuracy increases because the number of classified instances grows. When the model has already learned almost all instances the learning time grows linearly depending on the number of instances as the results showed, and thus the accuracy is one in practice. In this moment if a instance are misclassified then the neural network model have to learn this instances and the learning time is the largest as Fig. 8 top shows for one execution.

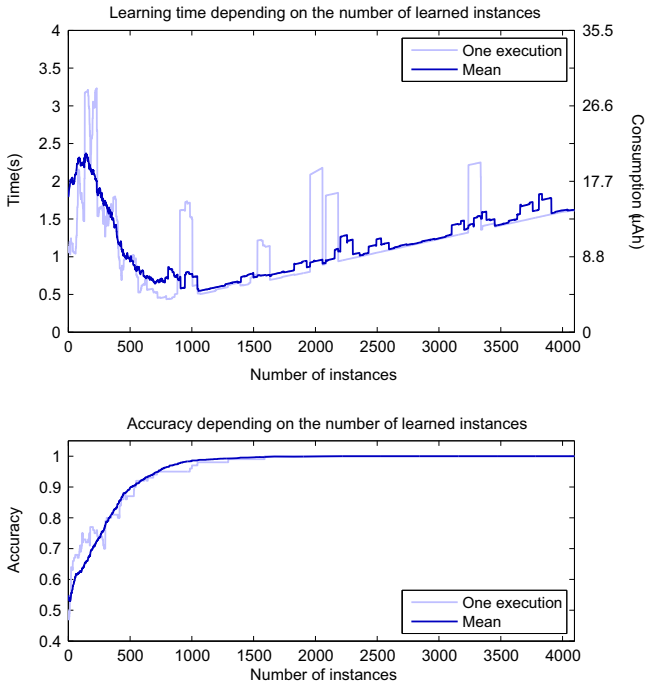

Fig. 8. Learning time, consumption (top graph) and accuracy (bottom graph) of a sensor/actuator used for an automatic watering system. (See text for details).

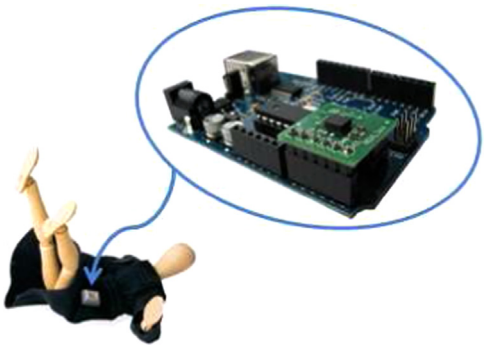

Fig. 9. A sensor based system attached to a person body for the implementation of a person fall detection system.

6.3. Fall detection system

Monitoring elderly or disabled people is in growing demand in modern societies, mainly because these people are dependent and unable to care for themselves. In particular, one important problem is to detect efficiently when a person falls. Different ways of addressing this problem have been proposed, like using a video surveillance systems, although if such system has the drawback of a strong sensitivity to light changes. A more efficient system to detect a person fall can be constructed using a sensor attached to the person body. The sensor, in this case, should include a 3-axis accelerometer together with a microcontroller that analyses the person relative inclination angle and movement, in order to detect abnormal situations, such as a strong downward movement or any sudden or violent displacement. Fig. 9 shows a schematic drawing of the described system.

Deducing the logic which describes the behaviour of the falls is a complex and very time consuming task. However, using a neural network based system simplifies enormously this task as a training process based on observed patterns can be implemented.

In such a system, the accelerometer senses the position of the device, acquiring the position in relationship to axes x, y, z. These coordinates are then sent to the microcontroller, in values normalised between 0 and 255 so they can be stored in a variable of "byte" type. A movement is considered a fall when in a short period of time (1 s), the position of the device changes from an initial state (vertical) to a final state (horizontal). Thus, a pattern to the system consists in the initial and final position of the movement plus the output that indicates whether the person has suffered or not a fall ($(x_0, y_0, z_0, x_1, y_1, z_1) \rightarrow fall?$). The number of patterns that can be stored on the system is delimited by Eq. (7), that in the present case leads to a maximum number of patterns to be stored equal to 167 patterns. As this number is quite small for such a complex problem, only patterns that a "supervisor" considered wrong will be used. For example, a pattern is sensed every second and the system determines whether or not a fall has occurred. If the supervisor (that during the training of the system can be the person carrying the device) resolves that the decision of the neural network model is wrong, this pattern is stored in memory and a new neural network model is calculated. The previous description is a modification to the standard C-Mantec algorithm that includes a more efficient storage of patterns for maximising the use of the limited memory resources of a microcontroller. For testing the system, a sensor node has been attached to a subject producing common movements such as sitting down, walking and stopping. The microcontroller has periodically gathered accelerometer data every second and 30,000 patterns have been obtained as data set.

Afterwards, the neural network based systems were checked, starting the process with no patterns in the memory, to only store one instance of the problem when the classification obtained does not match the supervisor output, that in our case, was the person wearing the system.

Fig. 10 top shows the time evolution of the system as the number of patterns presented increase. In the figure, the mean of

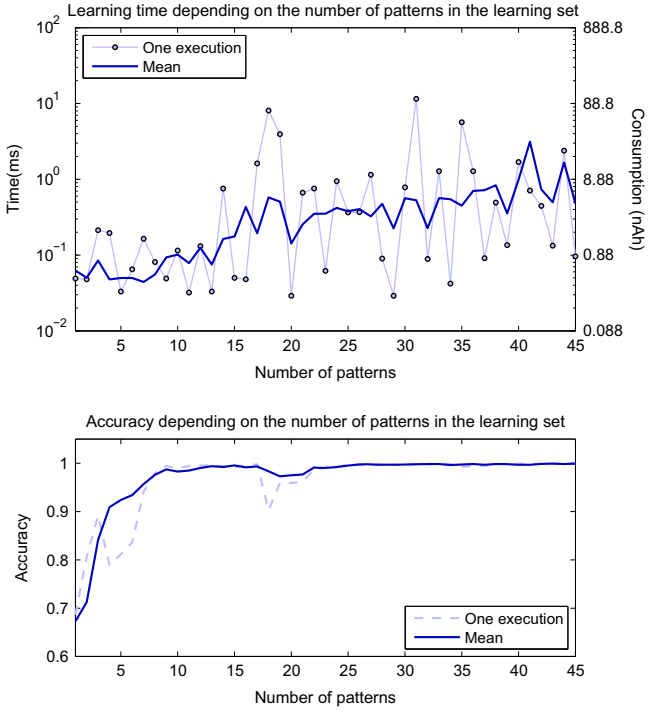

Fig. 10. Training time (top) and accuracy (bottom) of a fall detection system as a function of the number of observed patterns. Each graph includes average results over 30 independent samples, together with results corresponding to a single case.

30 executions is displayed together with a single case to observe the level of variability existing in the problem. Fig. 10 bottom shows the level of accuracy obtained as the patterns are presented to the system (the graph also includes the mean and a single observed case).

It is worth mentioning that the way this problem is treated is different from the previous two cases, as patterns are stored in the microcontroller memory only if they are misclassified.

7. Conclusions

The C-Mantec constructive neural network algorithm has been successfully implemented in a microcontroller board, adapting it to overcome the limitations imposed by the limited resources of memory and computing speed of the hardware device. The correct implementation of the algorithm has been verified in comparison to the original published results, obtaining that as the number of inputs is increased, the microcontroller implementation needs a small number of extra neurons and it is possible to observe a little reduction in performance accuracy due to rounding effects. Furthermore, a thorough comparison of the differences of using floating point or fixed precision representations has be carried out, concluding that better results can be obtained with an 8-bit fix precision representation leading to computation times approximately five times faster than using the standard floating point representation.

The implemented algorithm has been employed as a sensor/actuator system and applied to three case studies in order to demonstrate the efficiency and versatility of the resulting application. The three case studies chosen are problems defined in changing environments, and thus the decision-making of the sensor/actuator has to be adapted accordingly to the observed changes, thus needing a retraining of the neural network model that controls the decision process.

The observed reprogramming times are significantly low in the three case studies, being the energy consumption of the device also quite small, and even if a comparison to the traditional case in which the new code has to be transmitted from a central control unit has not been analysed, the results suggest a very important potential reduction.

As an overall conclusion, we have shown the suitability of C-Mantec for its application in a dynamic task using an Arduino UNO microcontroller. Nowadays, given the existence of devices with much more powerful computing resources than the considered board, the present study confirms the potential of the proposed algorithm for its application in real tasks where sensors/actuators are needed.

Acknowledgements

The authors acknowledge support from Junta de Andalucía through Grant Nos. P10-TIC-5770 and P08-TIC-04026, and from CICYT (Spain) through Grant No. TIN2010-16556 (all including FEDER funds).

References

Aiello, F., Bellifemine, F.L., Fortino, G., Galzarano, S., Gravina, R., 2011. An agent-based signal processing in-node environment for real-time human activity monitoring based on wireless body sensor networks. Eng. Appl. Artif. Intell. 24, 1147–1161.

Aleksendrić, D., Jakovljević, I., Irović, V., 2012. Intelligent control of braking process. Expert Syst. Appl. 39.

Angelov, P., 2010. Evolving Takagi–Sugeno fuzzy systems from data streams (eTS+). In: Angelov, P., Filev, D., Kasabov, N. (Eds.), Evolving Intelligent Systems. John Wiley and Sons and IEEE Press, IEEE Press Series in Computational Intelligence, pp. 21–50

Atmel, Datasheet 328. http://www.atmel.com/Images/doc8161.pdf.

Canete, E., Chen, J., Luque, R., Rubio, B., 2012. Neuralsens: a neural network based framework to allow dynamic adaptation in wireless sensor and actor networks. J. Netw. Comput. Appl. 35, 382–393.

Cela, A., Yebes, J.J., Arroyo, R., Bergasa, L.M., Barea, R., López, E., 2013. Complete low-cost implementation of a teleoperated control system for a humanoid robot. Sensors 13, 1385–1401.

Chakraborty, S., Ghosh, R., Ghosh, M., Fernandes, C.D., Charchar, M.J., Kelemu, S., 2004. Weather-based prediction of anthracnose severity using artificial neural network models. Plant Pathol. 53, 375–386.

Dhanalakshmi, P., Palanivel, S., Ramalingam, V., 2011. Pattern classification models for classifying and indexing audio signals. Eng. Appl. AI 24, 350–357.

Farooq, U., Amar, M., ulHaq, E., Asad, M.U., Atiq, H.M., 2010. Microcontroller based neural network controlled low cost autonomous vehicle. In: Proceedings of the 2010 Second International Conference on Machine Learning and Computing, IEEE Computer Society, Washington, DC, USA, pp. 96–100.

Franco, L., Elizondo, D., Jerez, J., 2009. Constructive Neural Networks. Springer-Verlag, Berlin

Frean, M., 1990. The upstart algorithm: a method for constructing and training feedforward neural networks. Neural Comput. 2, 198–209.

Gómez, I., Franco, L., Jerez, J., 2009. Neural network architecture selection: can function complexity help?. Neural Process. Lett. 30, 71–87.

Han, C.C., Kumar, R., Shea, R., Srivastava, M., 2005. Sensor network software update management: a survey. Int. J. Netw. Manag. 15, 283–294.

Haykin, S., 1994. Neural Networks: A Comprehensive Foundation. Prentice Hall

Huang, G.B., Saratchandran, P., Sundararajan, N., 2005. A generalized growing and pruning rbf (ggap-rbf) neural network for function approximation. IEEE Trans. Neural Netw. 16, 57–67.

Hunter, D., Hao, Y., Pukish, M., Kolbusz, J., Wilamowski, B., 2012. Selection of proper neural network sizes and architectures – a comparative study. IEEE Trans. Ind. Appl. 8, 228–240.

Kodogiannis, V.S., Boulougoura, M., Wadge, E., Lygouras, J.N., 2007. The usage of soft-computing methodologies in interpreting capsule endoscopy. Eng. Appl. AI 20, 539–553.

Kopetz, H., 1997. Real-Time Systems: Design Principles for Distributed Embedded Applications, 1st edition Kluwer Academic Publishers, Norwell, MA, USA

Kornuta, J.A., Nipper, M.E., Dixon, Brandon, 2013. Low-cost microcontroller platform for studying lymphatic biomechanics in vitro. J. Biomech. 46 (1), 183–186.

Lakshmi, K., Subadra, M., 2013. A survey on FPGA based MLP realization for on-chip learning. Int. J. Sci. Eng. Res., 1–9.

Lian, K.Y., Hsiao, S.J., Sung, W.T., 2013. Intelligent multi-sensor control system based on innovative technology integration via zigbee and WI-FI networks. J. Netw. Comput. Appl. 36, 756–767.

Lughofer, E., 2011. Evolving Fuzzy Systems – Methodologies, Advanced Concepts and Applications. Studies in Fuzziness and Soft Computing, vol. 266. Springer, http://dx.doi.org//10.1007/978-3-642-18087-3.

Luque-Baena, R.M., Urda, D., Subirats, J.L., Franco, L., Jerez, J.M., 2013. Analysis of cancer microarray data using constructive neural networks and genetic algorithms. In: Rojas, I., Guzman, F.M.O. (Eds.), Proceedings of the IWBBIO, International Work-Conference on Bioinformatics and Biomedical Engineering, pp. 55–63.

Mahmoud, S., Lotfi, A., Langensiepen, C., 2013. Behavioural pattern identification and prediction in intelligent environments. Appl. Soft Comput. 13, 1813–1822.

Mamdoohi, G., Fauzi Abas, A., Samsudin, K., Ibrahim, N.H., Hidayat, A., Mahdi, M.A., 2012. Implementation of genetic algorithm in an embedded microcontroller-based polarization control system. Eng. Appl. Artif. Intell. 25, 869–873.

Marwedel, P., 2006. Embedded System Design. Springer-Verlag, New York, Inc., Secaucus, NJ, USA

Ortega-Zamorano, F., Subirats, J.L., Jerez, J.M., Molina, I., Franco, L., 2013. Implementation of the C-Mantec neural network constructive algorithm in an arduino UNO microcontroller. Lect. Notes Comput. Sci. 7902, 80–87.

Oxer, J., Blemings, H., 2009. Practical Arduino: Cool Projects for Open Source Hardware. Apress, Berkeley, CA, USA

Park, K., Shin, H., 2013. Stock price prediction based on a complex interrelation network of economic factors. Eng. Appl. AI 26, 1550–1561.

Rassam, M., Zainal, A., Maarof, M., 2013. An adaptive and efficient dimension reduction model for multivariate wireless sensor networks applications. Appl. Soft Comput. 13, 1978–1996.

Sayed-Mouchaweh, M., Lughofer, E., 2012. Learning in Non-Stationary Environments: Methods and Applications. Springer, New York

Sengupta, S., Das, S., Nasir, M., Panigrahi, B.K., 2013. Multi-objective node deployment in WSNS: in search of an optimal trade-off among coverage, lifetime, energy consumption, and connectivity. Eng. Appl. Artif. Intell. 26, 405–416.

Shaikh, R., Thakare, V., Dharaskar, R., 2010. Efficient code dissemination reprogramming protocol for WSN. Int. J. Comput. Netw. Secur. 2, 116–122.

Subirats, J., Franco, L., Jerez, J., 2012. C-mantec: a novel constructive neural network algorithm incorporating competition between neurons. Neural Netw. 26, 130–140.

Subirats, J.L., Baena, R.M.L., Urda, D., Ortega-Zamorano, F., Jerez, J.M., Franco, L., 2013. Committee C-Mantec: a probabilistic constructive neural network. Lect. Notes Comput. Sci. 7902, 339–346.

Taylor, J.W., Buizza, R., 2002. Neural network load forecasting with weather ensemble predictions. IEEE Trans. Power Syst. 17, 626–632.

Urda, D., Canete, E., Subirats, J.L., Franco, L., Llopis, L., Jerez, J.M., 2012. Energy-efficient reprogramming in WSN using constructive neural networks. Int. J. Innov. Comput. Inf. Control 8, 7561–7578.

Urda, R.M., Luque, M.J., Jiménez, I., Turias, L. Franco, Jerez, J. M., 2013 A constructive neural network to predict pitting corrosion status of stainless steel. Lecture Notes in Computer Science, 7902, pp. 88-95, (2013). ISBN: 978-3-642-38678-7.
Wang, J., Xu, W., Gong, Y., 2010. Real-time driving danger-level prediction. Eng. Appl. Artif. Intell. 23, 1247–1254.
Wang, Q., Zhu, Y., Cheng, L., 2006. Reprogramming wireless sensor networks: challenges and approaches. Netw. IEEE 20, 48–55.
Yick, J., Mukherjee, B., Ghosal, D., 2008. Wireless sensor network survey. Comput. Netw. 52, 2292–2330.
Zhai, Y.J., Yu, D.L., 2009. Neural network model-based automotive engine air/fuel ratio control and robustness evaluation. Eng. Appl. Artif. Intell. 22, 171–180.