Implementation of the C-Mantec Neural Network Constructive Algorithm in an Arduino Uno Microcontroller

Francisco Ortega-Zamorano[1], José Luis Subirats[1], José Manuel Jerez[1], Ignacio Molina[2], and Leonardo Franco[1]

[1] Universidad de Málaga, Department of Computer Science, ETSI Informática, Spain
{fortega,jlsubirats,jja,lfranco}@lcc.uma.es
[2] Max Planck Institute, Munich, Germany
imol@uma.es

Abstract. A recently proposed constructive neural network algorithm, named C-Mantec, is fully implemented in a Arduino board. The C-Mantec algorithm generates very compact size neural architectures with good prediction abilities, and thus the board can be potentially used to learn on-site sensed data without needing to transmit information to a central control unit. An analysis of the more difficult steps of the implementation is detailed, and a test is carried out on a set of benchmark functions normally used in circuit design to show the correct functioning of the implementation.

Keywords: Constructive Neural Networks, Microcontroller, Arduino.

1 Introduction

Several technologies like Wireless Sensor Networks [1], Embedded Systems [2] and Real-time Systems [3] are nowadays being extensively used in all kind of industrial applications, most of which use microcontrollers [4] to implement. The recent advances in the computing power of this kind of systems are starting to permitting the use of learning systems, that are able to adjust its functioning as the input data is received, to manage the microcontrollers present in their structure. Neural networks [5] are a kind of flexible and widely used learning systems that are natural candidates for this task as they are very flexible. Nevertheless a disadvantage of neural networks is that learning needs intensive computing power and tends to be prohibitive even for modern systems. In this sense, a recently proposed neural network constructive algorithm has the advantage of being very fast in comparison to standard neural network training and further it creates very compact neural architectures that is useful given the limited memory resources of the microcontrollers.

In this work the C-Mantec[6] algorithm has been fully implemented in a microcontroller, as the training process is part of the software of the controller and it is not carried out externally as it is usually done. We have chosen the Arduino

I. Rojas, G. Joya, and J. Cabestany (Eds.): IWANN 2013, Part I, LNCS 7902, pp. 80–87, 2013.

UNO board [7] as it is a popular, economic and efficient open source single-board microcontroller. C-Mantec is a neural network constructive algorithm designed for supervised classification tasks. One of the critical factors at the time of the implementation of the C-Mantec algorithm is the limited resources of memory of the microcontroller used (32 KB Flash, 2KB RAM & 1KB EPROM memory) and in this sense the implementation has been done with integer arithmetic except for one of the parameters of the algorithm. The paper is structured as follows: we first, briefly describe the C-Mantec algorithm and the Arduino board, secondly we give details about the implementation of the algorithm, to finish with the results and the conclusions.

2 C-Mantec, Constructive Neural Network Algorithm

C-Mantec (Competitive Majority Network Trained by Error Correction) is a novel neural network constructive algorithm that utilizes competition between neurons and a modified perceptron learning rule (termal perceptron) to build compact architectures with good prediction capabilities. The novelty of C-Mantec is that the neurons compete for learning the new incoming data, and this process permits the creation of very compact neural architectures. The activation state (S) of the neurons in the single hidden layer depends on N input signals, ψ_i, and on the actual value of the N synaptic weights (ω_i) and the bias (b) as follows:

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

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

$$\phi = \sum_{i=1}^{N} \omega_i \, \psi_i - b \tag{2}$$

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:

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

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 to the standard perceptron learning rule is that the thermal perceptron incorporates the factor T_{fac}. 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) that is decreased as the learning process advances.

$$T_{fac} = \frac{T}{T_0} e^{-\frac{|\phi|}{T}} \tag{4}$$

C-Mantec, as a CNN algorithm, has in addition the advantage of generating online the topology of the network by adding new neurons during the training

phase, resulting in faster training times and more compact architectures. The C-Mantec algorithm has 3 parameters to be set at the time of starting the learning procedure. Several experiments have shown that the algorithm is very robust against changes of the parameter values and thus C-Mantec operates fairly well in a wide range of values. The three parameters of the algorithm to be set are:

- I_{max}: maximum number of iterations allowed for each neuron present in the hidden layer per learning cycle.
- g_{fac}: growing factor that determines when to stop a learning cycle and include a new neuron in the hidden layer.
- F_{itemp}: determines in which case an input example is considered as noise and removed from the training dataset according to the following condition:

$$\forall X \in \{X_1, X_2, ..., X_N\}, delete(X) \mid NTL \geq (\mu + F_{itemp} \cdot \sigma), \qquad (5)$$

where N represents the number of input patterns of the dataset, NTL is the number of times that the pattern X has been presented to the network on the current learning cycle, and the pair $\{\mu, \sigma\}$ corresponds to the mean and variance of the normal distribution that represents the number of times that each pattern of the dataset has been learned during the learning cycle. This learning procedure is essentially based on the idea that patterns are learned by those neurons, the thermal perceptrons in the hidden layer of the neural architecture, whose output differs from the target value (wrongly classified the input) and for which its internal temperature is higher than the set value of g_{fac}. In the case in which more than one thermal perceptron in the hidden layer satisfies these conditions at a given iteration, the perceptron with the highest temperature is the selected candidate to learn the incoming pattern. A new single neuron is added to the network when there is no thermal perceptron that complies with these conditions and a new learning cycle starts.

3 The Arduino UNO Board

The Arduino Uno is a popular open source single-board microcontroller based on the ATmega328 chip [8]. It has 14 digital input/output pins, which can be used as input or outputs, and in addition, has some pins for specialized functions, for example 6 digital pins can be used as PWM outputs. It also has 6 analog 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 (0.5 KB are used for the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM. 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 Uno board are 6.8 and 5.3 cm respectively, with the USB connector and power jack extending beyond the former dimension. A picture of the Arduino UNO board is shown in Fig. 1.

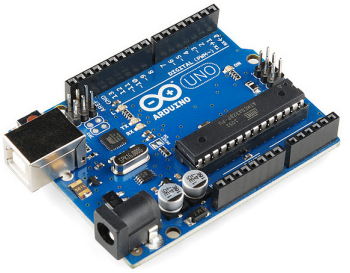

Fig. 1. Picture of an Arduino UNO board used for the implementation of the C-Mantec algorithm

4 Implementation of the C-Mantec Algorithm

The C-Mantec algorithm implemented in the wiring code is transferred by USB from the development framework from the PC to the board. The execution of the algorithm comprises two phases or states, because first, the patterns to be learnt have to be loaded into the EEPROM, and then the neural network learning process can begin. The microcontroller state is selected using a digital I/O pin. We explain next, the main technical issues considered for the implementation of the algorithm according to the two phases mentioned before:

4.1 Loading of Patterns

It is necessary to have the patterns stored in the memory board because the learning process work in cycles and use the pattern set repeateadly. The truth (output) value of a given Booelan pattern is stored in the memory position that corresponds to the input. For example, for the case of pattern of 8 inputs, the input pattern "01101001" that corresponds to the decimal number 105 and has a truth value of 0 would be stored by saving a value of 0 in the EEPROM memory position 105. The Arduino Uno EPROM has 1KB of memory, i.e., 8192 bits (2^{13}) and thus this limits the number of Boolean inputs to 13. For the case of using an incomplete truth table, the memory is divided into two parts, a first one to identify the pattern output and a second part to indicate its inclusion or not in the learning set. In this case, of an incomplete truth table, the maximum number of inputs is reduced to 12.

For the case of using real-valued patterns is necessary to know in advance the actual number of bits that will be used to represent each variable. If one byte is used to represent each variable then from the following equation the maximum number of input patterns permitted can be computed:

$$N_P \cdot N_I + N_P/8 \leq 1024\,, \tag{6}$$

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 as they become necessary, action that is not easily implemented in microcontroller, so we decided to set a value for the maximum number of permitted neurons, that will be stored in the SRAM memory. 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 following variables related to the neurons:

- T_{fac}: must be a variable of *float* type and occupies 4 bytes.
- Number of iterations: an integer value with a range between 1000 and 100000 iterations, so it must be of type *long*, 4 bytes.
- Synaptic weights: almost all calculatios are based on these variable, so to speed up the computations we choose *integer* types of 2 bytes long.

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

$$4 \cdot N_N + 2 \cdot N_N + 2 \cdot N_N \cdot (N_I + 1) \leqslant 1024, \tag{7}$$

where N_I is the number of inputs. For the maximum number of permitted inputs (13), the maximum number of neurons is 30. The computation of T_{fac} is done using a float data type because it requires an exponential operation that can be done only with this type of data, but as its computation involves other data types (integers) , a conversion must be done. To make this change without losing accuracy, we multiply the value of T_{fac} by 1000, leading to values in the range between 0 and 1000. When we convert to integer data type, precision is lost starting from the fourth digital number. Weights are of integer type in the range from -32768 to 32767, and as they are multiplied by the value of T_{fac}, we compensate this change by dividing them by 1000. When any synaptic weight value is greater than 30, or less than -30, all weights are divided by 2. This change does not affect at all the procedure of the network as neural network are invariant to this type of rescaling. To avoid the overflow of the integer data type, we apply the previous transformation whenever a synaptic weight reach the maximum or minimum permitted values. One very important thing in the implementation is the the execution time needed by the algorithm. In our case, this value depends strongly on the number of neurons actually used, as this time grows exponentially as a function of the number of used neurons. Fig 2 shows the execution time as a function of the neurons used in an architecture generated by the C-Mantec algorithm.

5 Results

We have tested the correct implementation of the C-Mantec algorihtm in the Arduino board by comparing the obtained results, in terms of the number of neurons generated and the generalization accuracy obtained, with those previously observed when using the PC implementation. The test is also carried out

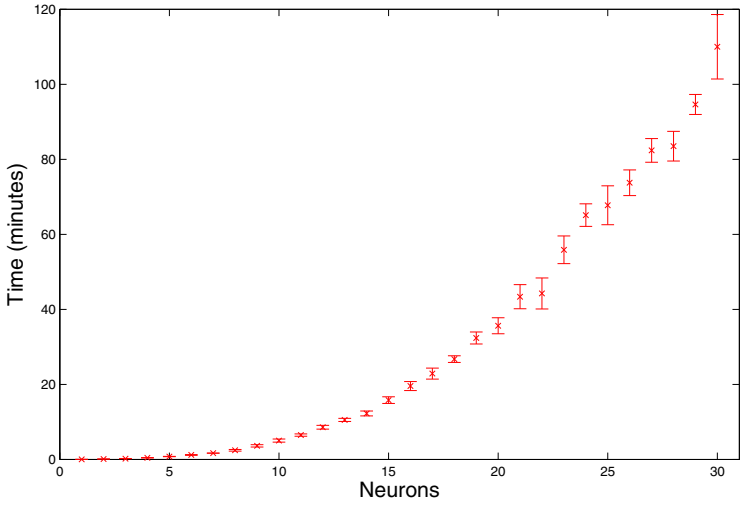

Fig. 2. Mean and standard deviation (indicated by error bars) of the execution time of the learning process as a function of the number of neurons used in a network created by the C-Mantec algorithm. The values shown are averages across 20 samples.

Table 1. Number of neurons and generalization ability obtained for a set of benchmark function for the implementation of the C-Mantec algorithm in an Arduino Uno board. (See text for more details).

Function	# Inputs	# Neurons		Accuracy generalization	
		Theory	Arduino	Theory	Arduino
cm82af	5	3,0±0,0	3,0±0,0	93,3±11,1	87,2±5,3
cm82ag	5	3,0±0,0	3,0±0,0	60.0±37,3	72,5±12,3
cm82ah	5	1,0±0,0	3,0±0,0	100.0±0,0	95,3±4,7
z4ml24	7	3,0±0,0	3,0±0,0	98,3±3,7	97,9±1, 1
z4ml25	7	3,1±0,9	3,1±0,9	90,8±12,3	86,0±0,9
z4ml26	7	3,0±0,0	3,0±0,0	96,7±5,9	94,6±0,4
z4ml27	7	3,0±0,0	3,0±0,0	99,2±2,8	99,9±0,9
9symml	9	3,0±0,0	3,0±0,0	99,4±0,9	97,5±1,2
alu2k	10	11,2±0,9	11,8±1,2	97,4±1,9	95,5±0,9
alu2l	10	18,9±1,5	19,3±1,3	79,2±5,5	70,3±1,3
alu2o	10	11,2±0,9	12,8±0,2	90,2±2,3	85,8±2,2

to analyze the effects of using a limited precision representation for the synaptic weights. A set of 10 single output Boolean functions from the MCNC benchmark were used to test the generalization ability of the C-Mantec algorithm. The C-Mantec algorithm was run with the following parameter values: $g_{fac} = 0.05$ and $I_{max} = 10000$. Table 1 shows the results obtained with the microcontroller for the

set of benchmark functions. The first two columns indicate the function reference name and its number of inputs. Third and fourth columns shows the number of neurons obtained by the PC and Arduino implementations, while fifth and last column shows the generalization ability obtained both for the PC and Arduino cases. The averages are computed from 20 samples and the standard deviation is indicated. The generalization ability shown in the table was computed using a ten-fold cross validation procedure.

6 Conclusion

We have successfully implemented the C-Mantec neural network constructive algorithm in an Arduino Uno board. The main issues at the time of the implementation are related to the memory limitations of the board. In this sense, we have analyzed the maximum number of Boolean and Real patterns that can be used for the learning process. For the case of Boolean patterns, we carried out a comparison against published results, showing that the algorithm works almost exact in comparison to the original PC implementation. As the number of inputs of the test functions increases, the Arduino implementation needs just a small extra number of neurons, and also a small degradation in the generalization accuracy is observed. These effects can be related to the limited numerical precision of the synaptic weights. The rounding effects should not in principle degrade the functioning of the algorithm, but affects the number of iterations needed to achieve convergence. Thus, we have also analyzed an important factor as it is the execution time of the algorithm. The results (cf. Figure 2) shows an exponential execution time increase as a function of the number of neurons in the constructed algorithms, and so for networks of approximately 15 neurons the execution time is around 20 minutes, while for 30 neurons this time increases up to two hours.

As a conclusion, and despite the previously mentioned limitations, we believe that the current implementation can be used in several practical applications, and we are planning to incorporate the C-Mantec algorithm in WSN in a near future.

Acknowledgements. 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. Yick, J., Mukherjee, B., Ghosal, D.: Wireless sensor network survey. Comput. Netw. 52(12), 2292–2330 (2008)
2. Marwedel, P.: Embedded System Design. Springer-Verlag New York, Inc., Secaucus (2006)
3. Kopetz, H.: Real-Time Systems: Design Principles for Distributed Embedded Applications, 1st edn. Kluwer Academic Publishers, Norwell (1997)

4. Andersson, A.: An Extensible Microcontroller and Programming Environment. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science (2003)
5. Haykin, S.: Neural networks: a comprehensive foundation. Prentice Hall (1994)
6. 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)
7. Oxer, J., Blemings, H.: Practical Arduino: Cool Projects for Open Source Hardware. Apress, Berkely (2009)
8. Atmel: Datasheet 328, `http://www.atmel.com/Images/doc8161.pdf`