Mathware & Soft Computing 3 (1996) 435-446

Learning under Hardware Restrictions in CMOS Fuzzy Controllers able to Extract Rules from Examples

F. Vidal-Verdú**, R. Navas** and A. Rodríguez-Vázquez*
*Dept. of Analog and Mixed-Signal Circuit Design
Centro Nacional de Microelectrónica-Univ. de Sevilla
Edificio CICA, C/Tarfia s/n, 41012-Sevilla, Spain
e-mail: angel@cnm.us.es
**Dpto. de Electrónica. Universidad de Málaga
Complejo Tecnológico, Campus de Teatinos, Málaga, Spain
e-mail: vidal@tecma1.ctima.uma.es

Abstract

Fuzzy controllers are able to incorporate knowledge expressed in if-then rules. These rules are given by experts or skilful operators. Problems arise when there are not experts or/and rules are not easy to find. Authors' proposal consists in an analog fuzzy controller which accepts structured language as well as input/output data pairs, thus rules can be extracted or tuned from human or software controller operation. Learning from data pairs has to be carried out under hardware restrictions in linearity, range and resolution. In this paper, modelling of building blocks arranged in a neuro-fuzzy architecture is made and issues related to on-chip learning are discussed. Computer simulations show that learning is possible for resolutions up to 6 bits, affordable with the cheapest VLSI technologies.

1 Introduction

A conventional fuzzy inference system realizes a fixed set of values with membership functions of predefined shapes and locations. Consequently:

- Fuzzy products based on these conventional approaches are unable to adapt their operation to changing environments. Due to this, they are commonly designed to cope with average environmental conditions, meaning that their performance will not be optimum for the majority of practical situations [1].
- Even if the lack of adaptability is tolerable, the formulation of a proper set of fuzzy rules able to cope with these average conditions remains a difficult task that requires a priori full knowledge about the system operation [2].

Neuro-fuzzy systems are intended to overcome these problems through the incorporation of learning to the conventional fuzzy inference procedure. Thus, a priori knowledge (i.e., fuzzy rules) is not required, but evolve as a result of the system operation in the field. Also, since neuro-fuzzy systems are intrinsically adaptive they are able to track to changing environments.

Although different authors have considered the realization of neuro-fuzzy systems in the form of software, these realizations cannot cope with the speed/power requirements of many practical applications, which motivates searching for dedicated hardware implementations [3]. One methodology to develop this kind of hardware implementations have been presented by the authors in [4]. To guarantee hardware simplicity and, hence, maximum operation speed, this methodology does not encompass detailed compensation of the spurious nonlinearities in their constitutive building blocks. Since these nonlinearities are corrected through the learning process, this does not pose a major limitation for practical use, similar to what is reported for neural network implementations in [5], [6]. However, to guarantee maximum yield in production of neuro-fuzzy silicon chips, this feature must be guaranteed before the chips are sent to production, what can only be done through the use of proper models, able to capture the non-linear behaviors, during the chip design phase. On the other hand, our effort focuses in hardware oriented algorithms, to allow future implementation of on-chip learning. Development of hardware models, proposal of hardware-oriented algorithms, and demonstration of learning based on both is our major concern in this communication.

2 Architecture and Building Blocks

Fig.1(a) shows the architecture of a neuro-fuzzy controller based on ANFIS [7], where building blocks are arranged in layers related to fuzzy algorithm steps. First layer performs fuzzification and each block s_{ij} implements a membership function. Second layer realizes t-norm through the minimum operator and the three remaining layers make a weighted average which involves singleton weighting and defuzzification. Electrical aspects in Fig.1(a) are discussed in [4]. Building blocks which belong to an only rule are enclosed by a dash line in Fig.1(a), thus we can note that membership functions are not shared by different rules. From the hardware point of view, sharing of membership functions among rules reports advantages in terms of area and power consumption, but only allows grid partitions of the universe of discourse, that is, the partition in each separate dimension or input determines the partition in the overall multidimensional input space, thus flexibility is reduced. On the contrary, we can define partitions directly on the multidimensional input space, thus generating scatter or tree partitions [7], by implementing one membership function for each label in each rule of the knowledge base.

This kind of redundancy in membership function generation improves locality, because rule antecedent output depends only on parameters related to a specific rule. Since rule antecedent output can be seen as a multidimensional membership function, or as a basis function in the interpolation context, this means that we generate basis functions which can be tuned independently, thus changes in a parameter through learning only affect the local piece of input space associated to the rule. Locality improves convergence and speed of learning, but is not an essential requisite. However, the less locality we have the more difficult convergence is, thus we need complex learning algorithms to reach the aim. Since we are interested in simple learning algorithms suitable for hardware implementation, sharing of membership functions is rejected in favor of redundancy.

Figure 1: Architecture and building blocks of a singleton fuzzy chip: architecture (a); membership function circuit (inset: generated bell-like membership function) (b); minimum circuit (c); normalization circuit (d) and singleton weighting circuit (e).

3 Modelling of Building Blocks

To demonstrate viability of architectures and learning algorithms in the hardware context, modelling of building blocks has to be made to carry out computer simulations. This section shows models for all building blocks in Fig.1. Modelling is realized here by finding mathematical expressions of input-output transfer functions of building blocks in Fig.1 with regard to first order equations of MOS transistors. Note that control transfer function is determined by the shapes and locations of membership functions in the first layer of Fig.1(a) and by singleton values in the fourth layer, while remaining layers contain blocks that do not have associated parameters the transfer function depends on. The former adaptive blocks are marked with an arrow in Fig.1(a) to differ from the latter fixed blocks.

Layer I: membership function circuitry

Membership functions are implemented in [4] using two differential pairs, as Fig.1(b) shows. This configuration enables independent tuning of width and location, which are linearly related to voltages E_{1ij} and E_{2ij}. On the other hand, independent tuning of the slope is achieved by using the compound transistors of Fig.2. The tuning parameter B is a voltage for compound transistors of Fig.2(b) and Fig.2(c), while it represents the decimal number coded in the digital word at gates of current switches in the compound transistor of Fig.2(a). The bell shape in the inset of Fig.1(b) is obtained by adding the two outputs of the differential pairs, thus analytical modelling reduces to make this operation with the differential pair output expression based on a square law model of MOS transistor [8]. This expression for a differential pair with simple transistor is the following:

$$i_{1,2} \begin{cases} \sqrt{2\beta I_Q} x_d \sqrt{1 - x_d^2 \beta/(2I_Q)} & |x_d| \leq \sqrt{I_Q/\beta} \\ I_Q \mathrm{sgn} x_d & |x_d| \geq \sqrt{I_Q/\beta} \end{cases} \tag{1}$$

where x_d is the differential voltage input, I_Q is the differential pair bias current and β is the large signal transconductance factor of transistors. The resulting model is a piecewise one, where each piece is obtained by summing different parts of differential pair input-output characteristic. Special care is necessary to model shape evolution when voltages E_{1ij} and E_{2ij} get closer and closer.

Figure 2: Compound transistors to control membership function slope and singleton values: digital transistor (a); parallel transistor (b) and series transistor (c).

Equation (1) is also valid for the digital transistor in Fig.2(a). More complex expressions are obtained for the series and parallel configurations. In particular, the differential pair obtained with parallel transistor in Fig.2(b) obeys the following equations:

$$i_{1,2} \approx \left[\begin{array}{ll} \sqrt{2(\beta_1+\beta_2)\left(I_Q - \frac{2\beta_1\beta_2}{\beta_1+\beta_2}B^2\right)}x_d \sqrt{1 - x_d^2 \frac{(\beta_1+\beta_2)}{2\left(I_Q - \frac{2\beta_1\beta_2}{\beta_1+\beta_2}B^2\right)}} & |x_d| \leq l_1 \\ \mathrm{sgn} x_d \Big(I_Q - \frac{c}{\beta_1+\beta_2}\left(I_Q - \frac{\beta_1\beta_2}{\beta_1+\beta_2}B^2\right) + c\left(1 - \frac{c}{\beta_1}\right)\tilde{x}_d + & |x_d| \geq l_1 \\ + \frac{c}{\beta_1}\tilde{x}_d \sqrt{c(c-2\beta_1)\tilde{x}_d^2 + 2\frac{c\beta_1}{\beta_1+\beta_2}\left(I_Q - \frac{\beta_1\beta_2}{\beta_1+\beta_2}B^2\right)} \Big\} & |x_d| \leq l_2 \\ I_Q \mathrm{sgn}\ x_d & |x_d| \geq l_2 \end{array} \right.$$

$$\tilde{x}_d = x_d - \frac{\beta_2}{\beta_1+\beta_2}B \qquad c = \frac{2\beta_1(\beta_1+\beta_2)}{2\beta_1+\beta_2} \tag{2}$$

$$l_1 = \sqrt{\frac{\left(I_Q - \frac{2\beta_1\beta_2}{\beta_1+\beta_2}B^2\right)}{(\beta_1+\beta_2)}} \qquad l_2 = \frac{\beta_2 B}{\beta_1+\beta_2} + \sqrt{\frac{I_Q - \frac{\beta_1\beta_2}{\beta_1+\beta_2}B^2}{\beta_1+\beta_2}}$$

The bell-shaped membership function is obtained again by adding outputs of both differential pairs in the circuit.

Solving differential pair equations for series transistors is even harder, because bottom transistor in Fig.2(c) works in ohmic region. However, a convenient model can be obtained based on proper understanding of the *modus operandi* of a series transistor. HSPICE simulations show that membership function shape with series transistors is quite similar to that obtained with simple transistors. If we make equal the expression of the slope for zero differential input of a simple differential pair and the expression of the slope for the differential pair with series transistor (which coincides with g_{ms} in Fig.2(c)), we obtain the large signal gain β_{eff} of transistors which would produce the "same" series transistor output in the membership function circuit,

$$\beta_{eff} = \frac{g_{ms}^2}{2I_Q} \tag{3}$$

Thus, equation (1) would remain valid for series transistor with β_{eff} as large signal gain factor of MOS transistors. HSPICE measurements show errors below 1.5% when we use this approximation for $B = 0V$.

Layer II: minimum circuit

Proposed multiple input minimum circuit in [4] is depicted in Fig.1(c). Modelling of circuit operation is straight by implementing minimum function in software.

Layer III: normalization circuit

Square-law calculations give the following expression for normalization circuit in [4]:

$$w_i^* = \frac{\beta_t}{\beta_b} w_i \left[1 + \frac{\sum\limits_{i=1,N} \sqrt{w_i}}{N\sqrt{w_i}} \left(\sqrt{1 + \frac{N\left(\left(\frac{\beta_b}{\beta_t} I_{SS} \right) - \sum\limits_{i=1,N} w_i \right)}{\left(\sum\limits_{i=1,N} \sqrt{w_i} \right)^2}} - 1 \right) \right]^2 \tag{4}$$

where β_t and β_b are the large signal transconductance factor of transistors M_{ti} and M_{bi} $(i = 1, ...N)$ in Fig.1(d). This circuit can provide an offset at output – needed for demanding speed requirements. However, simulations show that this offset has two non desired effects from learning point of view. First, since its elimination requires negative singletons, the singleton weighting circuitry becomes more complex. Second, since each singleton contributes to the whole universe of discourse, locality is lost. For simulations in the section V, the offset was eliminated at the core of each fuzzy set by different sizing of input and output transistors of the circuit, that is by making $\beta_t > \beta_b$ in (4). Other methods eliminate this offset at circuit level while preserving dynamic response.

Layer IV: singleton weighting

Singleton weighting at layer IV in Fig.1(a) is made by asymmetrical current mirrors. Fig.3 shows equations obtained with a square-law model of MOS transistors for current mirrors with compound transistors in Fig.2. Again, expression for series transistor is approximated, we have supposed gate voltage of bottom transistor much larger than drain voltage.

Figure 3: Adaptive current mirrors: (a) with series transistor; (b) with parallel transistor; (c) with digital transistor.

Layer V: aggregation

Aggregation of the rule consequents at output is realized in [4] by Kirchhoff current law, taking advantage from current output of rules. Obviously, modelling is direct in software by simple addition.

4 Learning Issues

Once hardware building blocks models have been developed, it is necessary to propose learning algorithms as well as a training strategy. Since our interest focuses on hardware implementations, we first propose hardware-oriented learning algorithms, suitable for hardware implementation, then we will see some guidelines to choose learning parameters involved in them.

Learning algorithms

Parameters updated through learning are those in the adaptive blocks of Fig.1(a), that is,

- parameters related to the *rule antecedent,* which are the set of voltages $\{E_{1ij}, E_{2ij}\}$, with $i = 1, ...N$ and $j = 1, ...M$, and the set of membership functions slopes at crossover points ($E_{1ij} = E_{2ij}$) S_{ij}, whose expressions coincide with transconductances given in Fig.2.
- parameters related to the *rule consequent*, which are the set of voltages (or digital words in Fig.3(c)) B_i, with $i = 1, ...N$, that determine gains of adaptive current mirrors in Fig.3 and thus singleton values.

Our proposal is based on an hybrid algorithm by Jang which updates parameters in the rule antecedent by mean of *backpropagation* rule and uses *least mean squares* for rule consequents training. Since *backpropagation* is costly to implement and requires derivability of every node in the system, *weight perturbation* [9] is proposed instead of *backpropagation* while singleton values are found with the *outstar rule* [10]. This translation provides a very simple hybrid algorithm that does not require derivability of transfer function nodes, thus performing the training with low resolution requirements.

Weight perturbation substitutes derivatives for finite differences and calculates the influence of each parameter on the global error, thus avoiding feedback paths. If ω is the learning parameter and $\zeta(\bullet)$ the global error at output, updating of w is given by

$$\Delta\omega = \frac{-\eta\,[\zeta(\omega) - \zeta(\omega + pert)]}{pert} = G(pert)\,(\zeta(\omega) - \zeta(\omega + pert)] \qquad (5)$$

where *pert* is a small perturbation, η is the learning rate, and both are constants. Note that weight update hardware involves evaluation of the error with perturbed and unperturbed weight and then multiplication by a constant $G(pert) = -\eta/pert$.

We use this strategy for the membership functions, thus ω in (5) is any parameter in the set $\{E_{1ij}, E_{2ij}, S_{ij}\}$ with $i = 1, ...N$ and $j = 1, ...M$. With regard to the singletons, it exploits the similarities of singleton fuzzy inference with the *counterpropagation* network. This becomes evident when one uses crisp rather than fuzzy sets, for the input labels. Based on this, our learning algorithm uses the outstar rule,

$$y^*_{i\mathrm{new}} = y^*_{i\mathrm{old}} + \mu[T - y(x)] \tag{6}$$

where T is the target output and μ is the learning rate; y^*_i is the singleton whose rule antecedent is maximum, that is $w^*_i(x) = \max\{w^*_1(x), w^*_2(x), ...w^*_N(x)\}$. Since singletons are given by parameters B_i in our case, we use (6) to update these parameters. Actually, singleton values perform a linear weighting of the normalized rule antecedent output in Fig.1(a), but they are determined by current mirrors gains in Fig.3(a) and Fig.3(b), which are not linearly related to parameters B_i. Thus updating B_i through (6) in these cases is only an approximation to the *outstar* rule.

Learning Parameters choice criteria

Like all gradient descendent algorithms, *weight perturbation* is quite sensitive to learning parameters. Large perturbation and learning rate values increase learning speed, but difficult convergence. On the other hand, small values make system to converge slowly [11]. *Outstar* shows a similar behavior, because can be understood like a gradient descendent algorithm. Apart from these considerations, some more specific criteria are desired to choose learning parameters. In hardware implementations, this choice is conditioned by resolution [11]. Criteria in section V have been first the choice of a quite small value of perturbation, whose minimum is the minimum step size applicable to learning parameter. On the other hand, since *weight perturbation* implements an approximate derivative, stability at minimum error point is intrinsically avoided by perturbation method, which forces little oscillations around this minimum when convergence succeeds, except when a limited resolution paralyzes the error evolution. Once a small perturbation and a desired maximum output error variation have been chosen, the learning rate is determined attending to the fact that the minimum parameter change is limited by resolution of learning parameters updated in (5), that is:

$$\eta \geq \frac{pert \times \Delta\omega_{\min}}{\left|[\zeta(\omega) - \zeta(\omega + pert)]\right|} \tag{7}$$

The learning rate in outstar is also determined by the same procedure,

$$\mu \geq \frac{\Delta y^*_{i_{\min}}}{\left|[T - y(x)]\right|} \tag{8}$$

where the desired maximum error at output is other parameter needed to determine μ.

5 Results

In this section some results are presented, for a neuro-fuzzy system with the architecture of Fig.1(a). The target function from which training data were obtained is $2.5 + \sin(\pi x)\sin(\pi y)$. Fig.4(a) shows this function in the interval $[0, 2] \times [0, 2]$, while Fig.4(b) shows the resulting surface of a 36-rules neuro-fuzzy system with ideal trapezoidal membership functions after being trained with 111 data pairs. The system was initialized with membership functions uniformly distributed along the universe of discourse while singletons were made all equal to the mean function value. Final RMSE error was bellow 2.5%. Further simulations were made in the interval $[0, 1] \times [0, 1]$, for a nine-rules controller and 36 data pairs, which speed up simulations without loose of generality, thanks to the surface symmetry.

Figure 4: Results of learning process: original surface (a) and learned surface (b) with first order models of a cotnroller and the hybrid algorithm; RMSE curves for weight perturbation (c) and hybrid algorithm (d) with software models of involved hardware.

Models of section III can be inserted in the architecture of Fig.1(a). The three options presented for membership functions and singletons result in many alternative combinations, of which only five are considered here. One uses digital transistors for membership functions and singleton. The rest cover the four possible combinations between the remaining composite transistors: parallel transistors in membership function and series in singleton (*parser*); series transistor in membership function and parallel in singleton (*serpar*); and parallel or series in both membership function and singletons (*parpar* and *serser*, respectively).

Several simulations were made for the five possibilities of previous paragraph, for different resolution and learning step sizes, as well as for hybrid (weight perturbation for rule antecedent and outstar for rule consequent) and weight perturbation (for both rule antecedent and rule consequent) algorithms. Building Blocks were designed to cover the same range, which was one to four for singleton values, one to two for membership function slopes, and -0.5 to 1.5 for parameters E_{kij} (see Fig.1(b)). Fig.4(c) and Fig.4(d) show RMSE curves from these simulations, where learning parameters were chosen under criteria in section IV. For the weight perturbation algorithm (Fig.4(c)), these parameters were $pert = 0.03$, $\eta/pert = 0.26$ (see equation (5)) for a resolution of 7 bits, for all combinations except for digital one, where parameters were scaled to $pert = 0.12$ and $\eta/pert = 0.5$ for the same resolution. With regard to the hybrid algorithm (Fig.4(d)), parameter μ, besides previous parameters for weight perturbation, was fixed to 0.01 for "analog" combinations, except for the combination with series transistors in both adaptive layers, where its value was 0.03. In the digital case, it was fixed to 0.1.

RMSE curves of Fig.4 shows a better behavior of hybrid algorithm as comparing to weight perturbation, which can also be observed in Table 1, where two performance parameters are given for both algorithms. The performance parameters are [11]:

- *the asymptotic* RMSE error $E_\infty = \lim_{n\to\infty} E_{\text{out}}(n)$, where n is the number of epochs, thus this parameter is the value of the output RMSE error after a sufficiently long training and
- *the convergence time* T_C, expressed in number of epochs from the beginning of learning up to the moment when E_{out} has been reduced below a given threshold. An epoch is defined as the application of the full training set to the network.

	asymptotic RMSE error	convergence time T_C
parpar_h	0.099	17
parpar_wp	-	-
parser_h	0.097	3
parser_wp	0.085	3
serpar_h	0.084	9
serpar_wp	0.090	77
serser_h	0.096	17
serser_wp	0.063	68
digital_h	0.087	7
digital_wp	0.096	30

Table 1: Learning Performance Parameters.

Apart from specific data, which can be improved by tuning of learning parameters, we can say in general that hybrid algorithm improves convergence time. The reason could be that this algorithm reinforces locality [12], by reinforcing rules with maximum influence in each training datum. Besides of this advantage, locality is an important issue in fuzzy systems, because it allows us to establish a correspondence between the singleton value of a rule and the function values in the area of the universe of discourse that is "occupied" by the rule. This "transparency" allows us to introduce knowledge in fuzzy systems, what is their major feature. Thus, a neuro-fuzzy system trained with hybrid algorithm will be initialized much better, and danger to fall in local minima minimized.

Finally, we can highlight the low resolution required for learning purposes, 7 bits in our simulations, what is affordable with even the cheapest VLSI technologies [13]. Learning is possible even for lower resolutions, as Fig.5(a) illustrates for 6 bits. On the other hand, learning rate μ for the outstar rule has been fixed to 0.01 in Fig.5(b), while resolution varies in a parpar simulation, showing influence of resolution in learning parameters.

Figure 5: Relationship between resolution and learning parameters.

References

[1] H. Takagi: "Applications of Neural Networks and Fuzzy Logic to Consumer Products". pp. 8-12 in *Fuzzy Logic Technologies and Applications,* New-York: IEEE Press 1994.

[2] T. Takagi and Sugeno: "Derivation of Fuzzy Control Rules from Human Operator's Control Action". *Proc. of the IFAC Symp. on Fuzzy Inf., Knowledge Representation and Decision Analysis,* pp. 55-60, July 1989.R.J. Marks II (ed.): *"Fuzzy Logic Technologies and Applications".* New-York: IEEE Press 1994.

[3] T. Yamakawa: "A Fuzzy Inference Engine in Nonlinear Analog Mode and Its Application to a Fuzzy Logic Control". *IEEE Trans. on Neural Networks,* Vol. 4, pp. 496-522, May 1993.

[4] F. Vidal-Verdú , A. Rodríguez-Vázquez, B. Linares-Barranco and E. Sánchez-Sinencio: "A Basic Building Block Approach to CMOS Design of Analog Neuro/Fuzzy Systems", *Proc. of the Third IEEE International Conference on Fuzzy Systems,* June 1994.

[5] P. W. Hollis and J. J. Paulos: "A Neural Network Learning Algorithm Tailored for VLSI Implementation", *IEEE Transactions on Neural Networks,* Vol. 5, No. 5, September 1994.

[6] J. B. Lont and W. Guggenbl. "Analog CMOS Implementation of a Multilayer Perceptron with Nonlinear Synapses", *Neural Networks,* Vol. 3, No. 3, pp. 457-465, May 1992.

[7] J.S.R. Jang and C.T. Sun: "ANFIS: Adaptive-Network-Based Fuzzy Inference System". *IEEE Trans. on Systems, Man and Cybernetics,* Vol. 23, pp. 665-685, May 1992.

[8] E.A. Vittoz: "Future of Analog in the VLSI Environment". *Proc. of IEEE ISCAS'90,* pp. 1372-1375, 1990.

[9] M. Jabri and B. Flower: "Weight Perturbation: An Optimal Architecture and Learning Technique for Analog VLSI Feedforward and Recurrent Multilayer Networks" *IEEE Transactions on Neural Networks,* Vol. 3, No. 1, pp. 154-157, January 1992.

[10] J. M. Zurada. *"Introduction to Artificial Neural Systems".* West Publishing 1992.

[11] L. M. Reyneri and Enrica Filippi: "An Analysis on the Performance of Silicon Implementations of Backpropagation Algorithms for Artificial Neural Networks". *IEEE Transactions on Computers,* Vol. 40, No. 12, December 1991.

[12] J. Nie and D. A. Linkens: "Learning Control Using Fuzzified Self-Organizing Radial Basis Function Network", *IEEE Transactions on Fuzzy Systems,* Vol. 1, No. 4, pp. 280-287, November 1993.

[13] M.J.M. Pelgrom et al.: "Matching Properties of MOS Transistors". *IEEE Journal of Solid-State Circuits,* Vol. 24, No. 5, Oct. 1989, pp. 1433-1440.