VLSI Implementation of a Fully Parallel Stochastic Neural Network

J.M.Quero, J.G.Ortega, C.L.Janer and L.G.Franquelo IEEE MEMBER

Dpto. de Ingeniería de Sistemas y Automática
Univ. de Sevilla, Spain

Abstract– In this paper we present a purely digital stochastic implementation of multilayer neural networks. We have developped this implementation using an architecture that permits the addition of a very large number of synaptic connections, provided that the neuron's transfer function is the hard limiting function. The expression that relates the design parameter, that is, the maximun pulse density, with the accuracy of the operations has been used as design criterium. The resulting circuit is easily configurable and expandable.

I. Introduction

Stochastic neural networks architectures have recently received much attention [1],[2]. The use of these architectures decreases the amount of hardware needed to realize the operations involved in neurons. If we look at the transfer function of a neuron ($y_i = \sigma(\sum_{j=1}^{j=n} \omega_{ij} y_j + I_i)$) we realize that we have to evaluate a set of arithmetic operation, more precisely, a set of products and additions. In order to make efficient electronic implementations of neural networks containing a high number of neurons, it would be desirable that these operations could be performed by simple circuitry.

Stochastic logic systems realize pseudoanalog operations using stochastically coded pulse sequences. These pulse sequences can be generated using a group of digital comparators. A digital codifications of variables are compared with uncorrelated random numbers producing uncorrelated stochastic signals whose values randomly take values 0 or 1. The average of these values can be viewed as an analog value in the range $[-a,a]$ or $[0,a]$, where $a = \frac{Rmax}{random_{max}} \leq 1$, with $Rmax$ the maximum value that can be stored in R_i and $random_{max}$ is the maximum random number generated.

Multiplication of two stochastic pulse sequences should produce another stochastic stream of pulses whose firing probability is the product of the input firing probabilities. This can be easily achieved if the input sequences are stochastically independent. The circuit that implements this operation is a simple AND gate.

Stochastic summation is a much more difficult operation to perform, specially if the terms to be added are signed. Two types of circuits have been described in the bibliography. One is the OR gate and the other is the up-down counter.

The up-down counters technique, although is widely used when implementing Hopfield's neural network, has a very important drawback. Pulses coming from other neurons have to be multiplexed in time (i.e. *serialized*) leading to *high computation times* if the network has many neurons and many connections per neuron. It should be also pointed out that the serialization that takes place is only efficient in neural networks whose neurons are fully connected .

If two pulse sequences are fed to an OR gate the output firing probability would be equal to the sum of both firing probabilities if the pulse sequences to be added did not overlap. This OR-based add function is thus distorted by pulse overlap. If only positive terms are to be added pulse overlap is not a drawback because it produces the non linear transfer function of a neuron. If signed inputs are to be added linear behavior is an essential feature. However this approach is extremely interesting due to the simplicity of the required hardware which would permit the digital implementation of arbitrary neural networks containing a very high number of neurons.

II. Proposed Stochastic Architecture

It has been proved that the OR gate based summation is extremely efficient if we restrict ourselves to neural networks in which neurons take only two discrete values. Due to the fact that the exponential function is monotonically increasing, and taking into account its properties, it can be deduced that $sign(\sum_{i=1}^{i=n} x_i) = sign(\prod_{i=1}^{i=n^-} e^{-x_i^-} - \prod_{i=1}^{i=n^+} e^{-x_i^+})$, where superscripts $+$ and $-$ are extended to positive and negative terms respectively.

. The last expression can be regarded as the comparison of two pulse streams generated by two stochastic multipliers, therefore no adder is needed. The only problem is to evaluate the exponential transformation in a efficient way.

If the neural network has been adimentionalized in such a way that all terms to be aggregated take values ranging from zero to a small number a close enough to zero, e^{-x_i} can be approximated by $1 - x_i$.

The resulting structure is shown in Fig.1. Input pulses are inverted by not gates and then fed to the corresponding AND gates by the first set of logic blocks. It should be pointed out that these logic blocks should maintain its output signal at high level while not being driven by the input pulses. The output logic block evaluates the neuron output. If only the "positive" pulse is at high level the counter is incremented by one and if the pulse is "negative" the counter is then decremented by one. The sign bit is changed if a zero crossing takes place. If a second order approximation of the exponential transformation was made, we could largely increase the value of a at the expenses of circuit's complexity (see Fig. 2).

Due to the fact that the exponential transformation is not evaluated exactly, it follows that the addition operation has a limited accuracy. If the total neuron's excitatory input and the total neuron's inhibitory input are very closed numbers, the neuron's output may be uncorrect. It has been proved that the accuracy of the addition is bounded by the following expression:

$$\hat{e} \leq \frac{e^{-a} + a - 1}{ae^{-a}} \simeq \frac{a}{2} \tag{1}$$

where $\hat{e}$ is the rate error of the addition operation. If the most accurate comparison that has to be evaluated is a known quantity, it follows that a has to be chosen so that $\hat{e} < \frac{DIFF}{VAL}$, where DIFF is the difference between the total positive neuron's input and the total negative neuron's input.

If the net inputs can take any values, the limited accuracy of the addition determines a region where the neuron's output will be uncertain.

III. Implementation

In this section several design issues, such us random number generation, network configuration, network expansion and hardware implementation, are described.

a) Random number generation

Linear Feedback Shift Register (LFSR) is one of the most studied digital techniques to generate pseudo-random numbers [4]. However, all the results given in previous sections are valid if pulse secuences are purely random -specially those involved in the same neuron. In our design we have used a 9-bit LFSR with $d_9 = XOR(q_0, q_4)$. We can obtain the cross-correlation (Crr) between shifted sequences $n(k)$ and $n(k+p)$, where $n(k)$ stands for the original normalized sequence and $n(k+p)$ is its p-steep ahead shift sequence. Fig 3 represents the cross-correlation between shifted random number sequences. It is clear that we need a 8-step shift at least to generate pulse sequences as uncorrelated as possible. These shifts have been made using 8-XOR gates. The number of uncorrelated pseudo-random sequences that can be simultaneously generated is $2^9/8 \simeq 64$. This number can be increased using a larger LFSR.

b) Network configuration

We have used pipelined parallel registers to allow neural network programmability with a minimun number of inputs. The size of these registers is related to the size of the LFSR and the maximum tolerated error in the exponential approximation. In this case we have choosen a 7-bit register for weight storage. If we consider input sequences to the network with a maximun probability of 0.5, the resulting multiplication has a mean of $\frac{2^7}{2^9}0.5 = 0.125$. According to Fig. 4, it leads to an exponential approximation error of 6% (linear approximation of the exponential transformation).

c) Network expansion

One of the most remarkable features of the proposed architecture is that the number of connections can be easily increased. Additions are calculated like products, so that sets of weights can be added using $\Psi+$ and $\Psi-$ signals without additional hardware (see Fig. 1). The number of hidden layers can also be increased with additional circuits in cascade.

d) Hardware implementation

The proposed stochastic architecture has been implemented using EDGE with ES2 1.5μ Standar Cell libraries. The resulting circuit contains two layers with five input unsigned pulse signals, five hidden neurons and one output neuron. Output neuron $\Psi+$ and $\Psi-$ signals are accessible, so that the number of hidden neurons may be increased.

IV. Results

In order to test the behaviour of the I.C., we have developped the controlled presented in [5]. This application consists on a perceptron that approximates a classification surface described by a 196-point array. A two layer perceptron with 10 hidden neurons have been configured using Back propagation. Two I.C.s have been used to implement this application.

All training input patterns are correctly classified by the network. The transient response of the output neuron for the input vector (0.0867,0.42,0.39) is shown in Fig. 5, where the clock rate is 7.5MHz.

Fig. 6 shows the behavior of the network as the input vector crosses the decision surface. In this figure the desired network's response and the actual network's response have been represented. Several reasons explain the slight differences between the two graphics: the discretization of the synaptic weights, the limited accuracy of the stochastic additions and the use of pseudorandom sequences instead of real random numbers.

V. Conclusions

In this paper we have presented a harware implementation of a multilayer neural network using the stochastic architecture that was proposed in [3]. This architecture can be easily expanded, and the circuit has been designed so that any multilayer network can be implemented by adding an appropriate number of I.C.s. As an example we have implemented the controller with perceptron-like structure which was presented in [5]. The response of the network matches with a high degree of accuracy the theoretical controller's response.

References

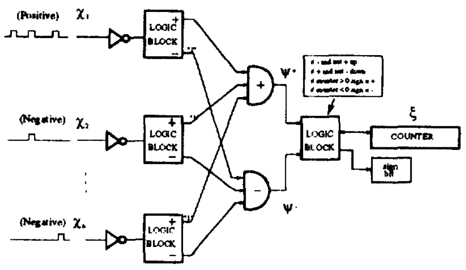
[1] Y. Kondo and Y. Sawada. Functional Abilities of a Stochastic Logic Neural Networks *IEEE Trans. on Neural Networks*, vol.3, pp.434-443, 1992.

[2] D.E. Van den Bout and T.K. Miller III. A Digital Architecture Employing Stochasticism for the Simulation of Hopfield Neural Nets. *IEEE Trans. on Circuit and Systems*, vol.36, pp. 732-738. 1989.

[3] C.L. Janer, J.M. Quero and L.G. Franquelo. Fully Parallel Summation in a New Stochastic Neural Network Architecture. *IEEE Int. Conf. on Neural Networks*, San Francisco, pp. 1498-1503. 1993.

[4] W. Peterson. Error Correcting Codes. MIT Press. 1992.

[5] J.M. Quero, J.M. Carrasco and L.G. Franquelo. Adaptative Energy Feed-Back Control for Resonant Converters Using Neural Networks. *23rd Annual Power Electronics Specialists Conference*, Toledo, Spain, pp. 800-806. 1992.



Maximun pulse density: a (a<<1)

Figure 1: Neuron's structure

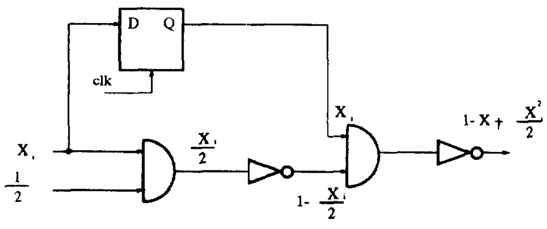


Figure 2: Evaluation circuit for the second order approximation of the exponential transformation.

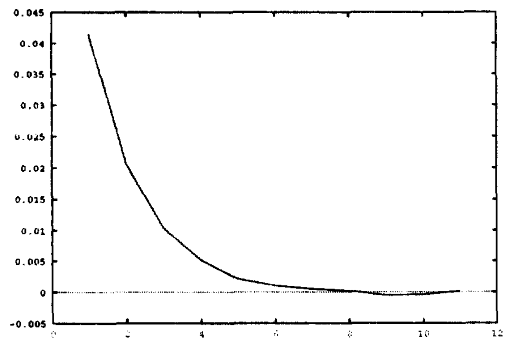


Figure 3: Cross-correlation between p-step ahead shift pseudo-random number sequences generated by a 9-bit LFSR)

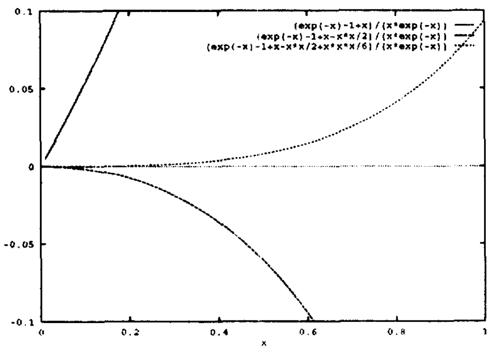


Figure 4: Normalized exponential transformation error, for the first, second and third order approximation.

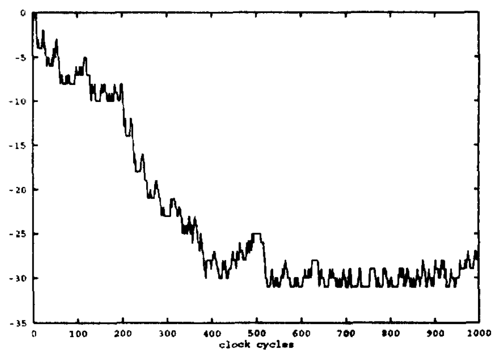


Figure 5: Transient response of the output neuron for the input vector (0.0867,0.42,0.39)

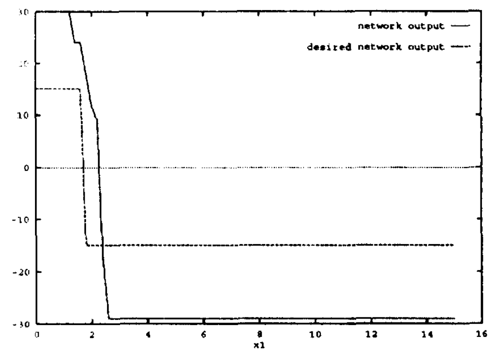


Figure 6: Network's desired and actual response as the input vector crosses the decision surface.