Probabilistic-based Neural Network Implementation

Josep L. Rosselló*, Vincent Canals, Antoni Morro
Electronic Technology Group, Physics Department
Universitat de les Illes Balears (U.I.B.)
Edifici Mateu Orfila, Cra. Valldemossa km. 7.5, 07122
Palma de Mallorca, Illes Balears, Spain
*E-mail address: j.rossello@uib.es.

Abstract— **This paper addresses a simple way for neural network hardware implementation based on probabilistic methodologies. We propose a new codification scheme that can be considered as an extension of stochastic computing (unipolar and bipolar codification formats), extending its representation range to any real number by using the ratio between two bipolar coded pulsed signals as codification method. Based on this codification, we propose the implementation of different linear and non-linear stochastic computational elements to be employed in artificial neural networks. Also this paper presents the accuracy associated to the proposed processing. The validation of the presented approach has been done with a sample application, (a spatial pattern classification example). The low cost in terms of hardware of the proposed methodology, along with the complexity of the mathematical expressions that can be implemented allows its use for massive parallel computing.**

Keywords--Artificial Neural Networks, stochastic arithmetic, stochastic computing, computational elements, parallel computation, pattern recognition

I. INTRODUCTION

Hardware implementation of neural networks is an attractive research topic in order to exploit the intrinsic parallel nature of neural systems. This fact allows hardware implementations (ad-hoc systems) to reach low computation times and energy if compared to software solutions. The implementation of neural networks for solving real problems as pattern recognition or signal processing requires the use of at least 10^3 neurons [1]. This number of neurons is not easy to integrate into present FPGA's or VLSI techniques [2] due the number of multiplier circuits required to evaluate the postsynaptic potential at each neuron. In consequence the implementation of large neural networks in hardware only is possible if the associated circuitry to the neural multipliers is reduced significantly. A feasible way to implement neural networks in hardware is the use of stochastic computing concepts [3-9]. With this computing scheme, probabilistic laws are applied to logic cells and stochastic pulse streams are used as analog variables [10-12]. The advantage of this kind of logic is the ability of reducing complex mathematical functions to simple digital circuitry. As an example, with the stochastic computing framework (unipolar codification) multiplication is performed using a single **AND** gate and the addition between two signals can be approximated using an **OR** gate. All this has led an intensive work devoted to implement Neural Networks by using stochastic computing concepts [3-9], in order to open the possibility to implement large neural systems in a single chip.

Neural Networks (NN) present many different applications in many fields as: classification [13-14], identification [15], associative memory and pattern recognition [16], biological applications [17], control [18], prediction and forecasting [19], optimization [20-21] and so on. Although all the advantages provided by stochastic computing, one of the shortcomings is that this logic is bounded between -1 and +1 (when using a bipolar codification), that represents a limitation for the special case of Neural Network design since the input data and the optimal weights values provided by traditional learning algorithms are not necessary bounded between these two values.

In this paper we show a new approach to neural network implementation by using an extension of the stochastic computing concept (Extended Stochastic Logic, ESL). With the proposed method, variables are defined as the ratio between two bipolar coded [2, 10] switching signals. The advantage is that any real number can be expressed by using this methodology and not only numbers between -1 and 1. We have to remark that the new notation simplifies some complex arithmetic blocks, (as the division) that are much more complex to implement with the traditional unipolar and bipolar stochastic computing paradigm.

II. PROBABILISTIC COMPUTING

To perform stochastic computations [10-12] a global clock provides the time interval during which all stochastic signals are evaluated (settled to 0 or +1). During a clock cycle each node presents a probability 'p' (or 1-p) of being in a high (low) state. This probabilistic-based coding provides a natural way of operating with analogue quantities (since commutation probabilities are defined between 0 and +1) using digital circuitry. When evaluated through logic gates, pulsed signals

This work was supported in part by the European Development Funds (FEDER) and in part by the Spanish Government under grant TEC2011-23113

follow probabilistic laws. As an example, the AND gate (in unipolar codification) provides at the output a signal with a switching probability equal to the product of their inputs (that is, the collision probability between signals).
Probabilities are defined between 0 and 1 and no negative numbers are allowed. To obtain negative numbers we must do a change of variables p*=2p-1 [2, 11] (where p is the switching probability of the signal). Therefore, since p is delimited between 0 and 1, p* would be bounded between -1 and +1. The reference (the zero value) is located at p=1/2. An advantage of this notation is that multiplication is still implemented by a single logic gate (now the XNOR gate [2]). Although this, the range is still bounded between two values (-1 and +1). Normalization could be a possible solution to this problem [2, 4, 22] (use of correct magnitudes for the signals such that their practical values are bounded between -1 and +1) but the problem is still present when implementing neural networks since the optimum weights for the connections can be out of the range of variation of signals, or the output of an linear or non-linear block (divisor, exponential,..) will be restricted to this range.
A new coding can be done using numbers represented by the ratio of the switching activity of different bipolar coded stochastic signals. With this notation data/values are fully represented by two values (the numerator and the denominator) rather than with a single stochastic signal
As happens with the traditional stochastic logic, binary numbers can be converted to stochastic signals using Random Number Generators (RNG) and comparators for the numerator and the denominator [3, 4, 23]. These blocks are defined as Binary to Pulse converters (B2P). Otherwise, to convert a switching signal to a binary quantity a counter operating during a certain period of time is needed. With coupled counters and a memory register we can implement the Pulse to Binary (P2B) converter [3, 23]. Therefore, combining two P2B blocks we can translate two stochastic signals (p,q) to binary numbers (P,Q).
In this new notation the conversion error depends on the mean value of the probability distributions associated with the numerator and denominator, similar as in the case of unipolar and bipolar encodings. The conversion error of the proposed notation is described by:

$$X^* = \frac{P^*}{Q^*} \rightarrow Error(X^*) = \left|\frac{1}{Q^*}\right| \cdot Error(P^*) + \left|\frac{P^*}{Q^{*2}}\right| \cdot Error(Q^*) \qquad (1)$$

With the proposed stochastic methodology, multiplication and division can be easily performed using **XNOR** gates (see the multiplier/divisor circuits in Fig. 1) between the two numerators and denominators of the quantities to be multiplied or divided.

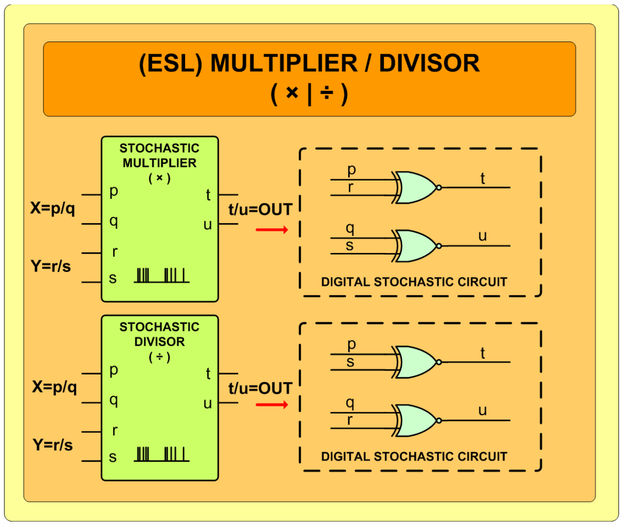

(ESL) MULTIPLIER / DIVISOR
(× | ÷)
STOCHASTIC MULTIPLIER (×)
X=p/q
Y=r/s
p
q
r
s
t
u
t/u=OUT
p
r
t
q
s
u
DIGITAL STOCHASTIC CIRCUIT
STOCHASTIC DIVISOR (÷)
X=p/q
Y=r/s
p
q
r
s
t
u
t/u=OUT
p
s
t
q
r
u
DIGITAL STOCHASTIC CIRCUIT

Fig. 1: Stochastic muliplier and divisor circuits used in the Extended Stochastic Logic (ESL). Where X=p*/q*, Y=r*/s* and Out=t*/u*

The addition and subtraction between two values can be done using the digital circuit shown in Fig. 2. When adding x*=*p*/q** and y*=*r*/s** the numerator of the sum (that we define as *t**) is performed with two **XNOR** gates (cross multiplication between numerator and denominator of the two numbers) and a multiplexer (that provide the mean value of the two products). Therefore the result in the numerator is *t*=(p*·s*+r*·q*)/2*. To obtain the denominator 'u*' we use a XNOR gate for multiplying signals '*q**' and 's*' and a multiplexer to divide the result between *2* (by performing the mean value between *q*·s** and *0*). Note the difference with traditional stochastic computing where only the mean of two values can be done instead of the real sum.

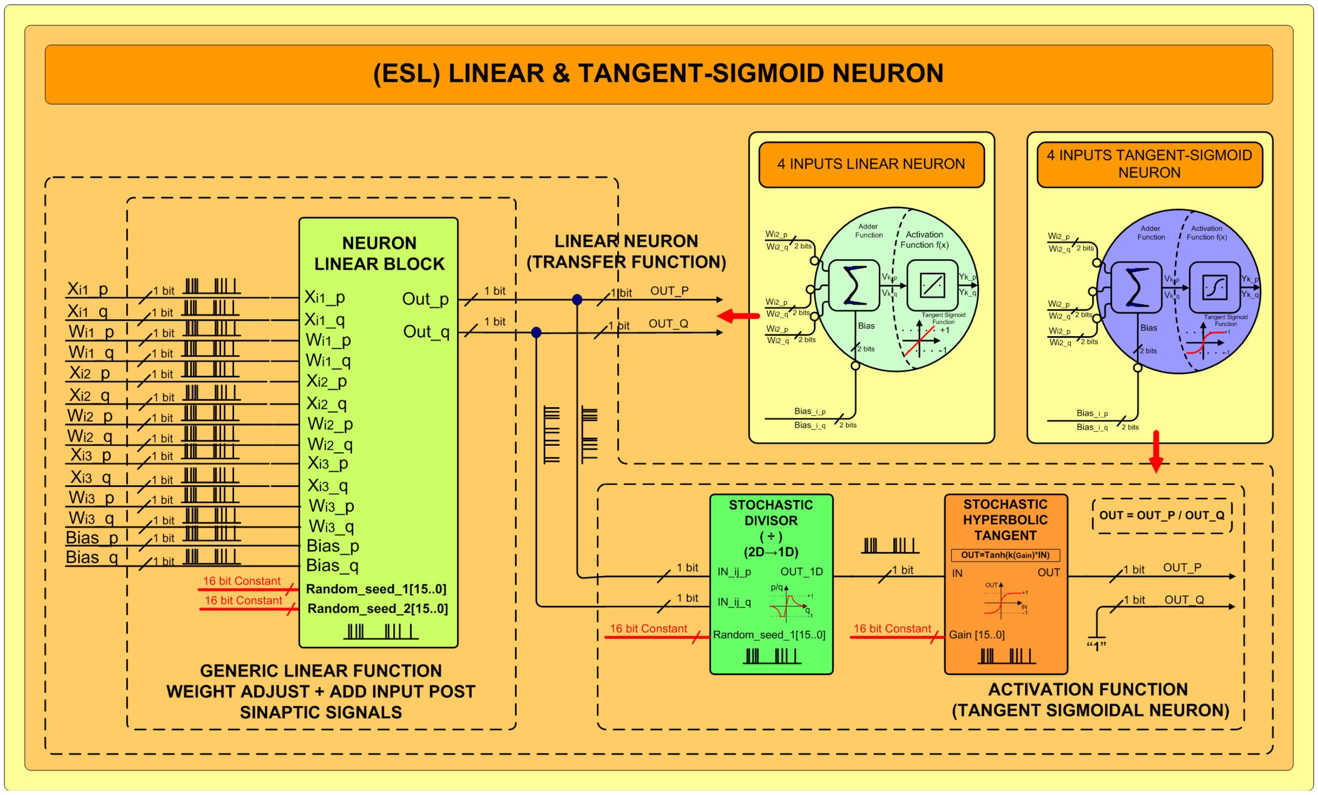

(ESL) LINEAR & TANGENT-SIGMOID NEURON
NEURON LINEAR BLOCK
LINEAR NEURON (TRANSFER FUNCTION)
4 INPUTS LINEAR NEURON
4 INPUTS TANGENT-SIGMOID NEURON
STOCHASTIC DIVISOR (÷) (2D→1D)
STOCHASTIC HYPERBOLIC TANGENT
OUT=Tanh(k(Gain)*IN)
OUT = OUT_P / OUT_Q
GENERIC LINEAR FUNCTION WEIGHT ADJUST + ADD INPUT POST SINAPTIC SIGNALS
ACTIVATION FUNCTION (TANGENT SIGMOIDAL NEURON)
16 bit Constant
Random_seed_1[15..0]
Random_seed_2[15..0]

Fig. 3: 3 inputs and 1 bias signal ESL neuron model (linear/tangent-Sigmoidal) digital block diagram

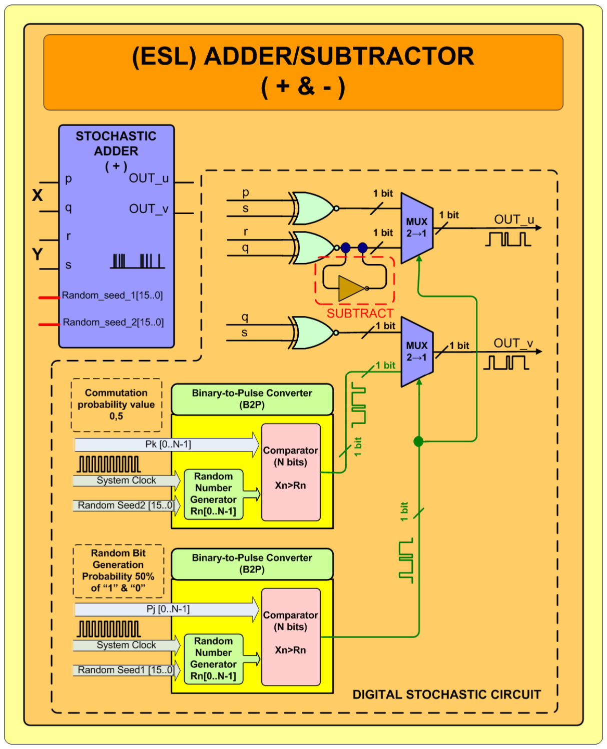

(ESL) ADDER/SUBTRACTOR (+ & -)
STOCHASTIC ADDER (+)
SUBTRACT
MUX 2→1
Binary-to-Pulse Converter (B2P)
Commutation probability value 0,5
Random Bit Generation Probability 50% of "1" & "0"
Comparator (N bits) Xn>Rn
Random Number Generator Rn[0..N-1]
System Clock
DIGITAL STOCHASTIC CIRCUIT

Fig. 2: Stochastic two values Adder / Substractor circuit in Extended Stochastic Logic (ESL). Where X=p*/q*, Y=r*/s* and Out=u*/v*

The ESL subtraction circuit is identical but including a NOT gate in order to change the sign of one of the two quantities. Therefore, the basic arithmetic functions (multiplication, division, addition and subtraction) can be obtained with the proposed method by using simple gates but with the additional advantage of the capacity to represent any real number. With this notation some functions as the division is much more simpler than with the traditional stochastic computing schemes.

III. Implementation of Artificial Neural Networks

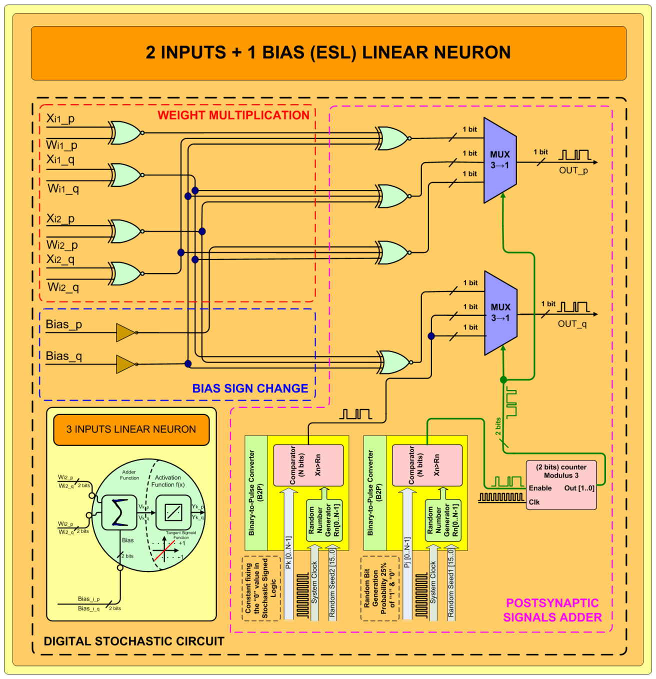
Using the basic concepts explained in the previous section, simple Artificial Neural Networks can be implemented in hardware. The output of each neuron (y_i) is related to the inputs (x_{ij}) following the next expression:

$$y_i = f(u_i) = f\left(\sum_{j=1}^{k} w_{ij}\cdot x_{ij} + b_i\right) = f\left(\sum_{j=0}^{k} w_{ij}\cdot x_{ij}\right) \qquad (2)$$

Where u_i represents the membrane potential of the i-th neuron, w_{ij} are the weighs and b_i is the bias level of the neuron. Parameter x_{ij} is the j-th input of the i-th neuron, where $x_{j0}=1$ and $w_{j0}=b_j$. The activation function "$f(x)$" depends on the type of neuron and can be the identity function (for linear neurons), a *Sign* function (in the case of Perceptron) or a more complex nonlinear function such as Log-Sigmoid or hyperbolic tangent.

In this paper we show how to implement Linear and Tangent-Sigmoidal neurons with the proposed ESL codification.
In Fig. 3 we show the implementation of a generic Stochastic Neuron that is composed by two parts, the linear part composed by multipliers (weight adjust) and an adder, and finally the activation function block (generally non-linear) which may consist in: a *Sign* function for the Perceptron or a Hyperbolic Tangent function in Tangent-Sigmoidal neurons.

Every neuron "i" would be composed by two linear elements: the first consists of "j" ESL multipliers responsible for the adjustment of each synaptic connection using weights w_{ij_p}/w_{ij_q}, and an adder block to create the signal u_i. In Fig. 4 we present the implementation of a linear neuron (Fig. 4) provided with two presynaptic inputs as well as a bias signal.



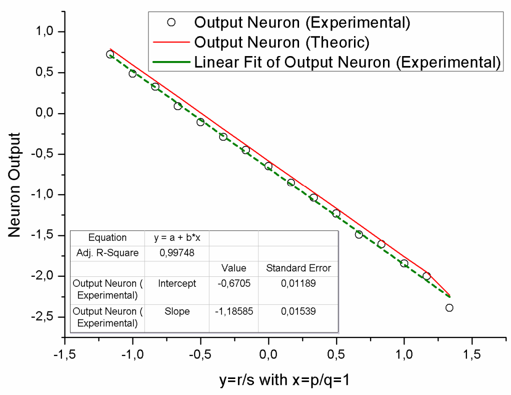
2 INPUTS + 1 BIAS (ESL) LINEAR NEURON
WEIGHT MULTIPLICATION
Xi1_p
Wi1_p
Xi1_q
Wi1_q
Xi2_p
Wi2_p
Xi2_q
Wi2_q
Bias_p
Bias_q
BIAS SIGN CHANGE
1 bit
MUX 3→1
OUT_p
OUT_q
2 bits
(2 bits) counter Modulus 3
Enable
Out [1..0]
Clk
3 INPUTS LINEAR NEURON
Adder Function
Activation Function f(x)
Bias
Binary-to-Pulse Converter (B2P)
Comparator (N bits)
Xn>Rn
Random Number Generator Rn[0..N-1]
Constant fixing the "0" value in Stochastic Signed Logic
Random Bit Generation Probability 33% of "1" & "0"
System Clock
Random Seed2 [15..0]
Random Seed1 [15..0]
POSTSYNAPTIC SIGNALS ADDER
DIGITAL STOCHASTIC CIRCUIT

Fig. 4: Three inputs (ESL) linear neuron implementation

On the left side of Fig.4 we find the multipliers of input signals, which are implemented through a pair a **XNOR** gates. The sign reversal of bias signal is carried by a pair of **NOT** gates. On the right side of Fig. 4 we find the sum block formed by two multiplexers ($3IN \rightarrow 1OUT$), one for the numerator and the other for the denominator. The selection of the channel is governed by a two-bit binary counter. The count enable of the counter is governed by a stochastic signal generator (B2P) (in order to maintain the randomness in the sum of components) with a switching activity of 1/3. Therefore the output of the linear neuron is governed by the following expression:

$$\begin{cases} \begin{cases} w_{i0} = \dfrac{w_{i0\_p}}{w_{i0\_q}} \\ w_{i1} = \dfrac{w_{i1\_p}}{w_{i1\_q}} \\ w_{i2} = \dfrac{w_{i2\_p}}{w_{i2\_q}} \end{cases} \rightarrow \begin{cases} x_{i1} = \dfrac{x_{i1\_p}}{x_{i1\_q}} \\ x_{i2} = \dfrac{x_{i2\_p}}{x_{i2\_q}} \end{cases} \rightarrow \begin{cases} a = (w_{i1\_q} \cdot x_{i1\_q}) \cdot (w_{i2\_q} \cdot x_{i2\_q}) \\ b = (w_{i0\_q}) \cdot (w_{i2\_q} \cdot x_{i2\_q}) \\ c = (w_{i0\_q}) \cdot (w_{i1\_q} \cdot x_{i1\_q}) \end{cases} \rightarrow \\ \begin{cases} OUT = u_i = \sum_{j=0}^{n-1} w_{ij} \cdot x_{ij} = \dfrac{u_{i\_p}}{u_{i\_q}} = \dfrac{\left(1/n\right) \cdot (w_{i0\_p} \cdot a + w_{i1\_p} \cdot b + w_{i2\_p} \cdot c)}{\left(1/n\right) \cdot \left((w_{i0\_q}) \cdot (w_{i1\_q} \cdot x_{i1\_q}) \cdot (w_{i2\_q} \cdot x_{i2\_q})\right)} \\ n = 3 \end{cases} \end{cases} \quad (3)$$

To test the correct operation of the linear neuron developed we have set the value of the input signals at "$x_{i1\_p}/x_{i1\_q} = +1$", bias "$w_{i0\_p}/w_{i0\_q} = -0.081067$", and the weights of the network input to "$w_{i1\_p}/w_{i1\_q} = -0.58052$, $w_{i2\_p}/w_{i2\_q} = -1.1766$". Varying the value of the input signal "x_{i2_p}/x_{i2_q}" between $\{-1.17, +1.33\}$ we obtain the results of Fig. 4 that fit perfectly with those theoretically predicted.



Equation	y = a + b*x		
Adj. R-Square	0,99748		
		Value	Standard Error
Output Neuron (Experimental)	Intercept	-0,6705	0,01189
Output Neuron (Experimental)	Slope	-1,18585	0,01539

Fig. 5: Linear neuron experimental measurements (symbols) vs. expected behavior (solid line)

Finally, we show the implementation of a tangent sigmoidal neuron, which consists of a linear block (described above) and a tangent sigmoidal activation function. The activation function (Fig. 3) is composed by a traditional stochastic divider (a description of this block is found in Fig. 6), that provides an estimation of the ratio *p/q*. The output of the divider is the input to the block dedicated to implement the Hyperbolic Tangent block (Fig. 7).
The input to the block shown in Fig. 7 (signal "IN" in the Hyperbolic Tangent block) must be a single switching signal equal to the ratio of the numerator and denominator. Since quantities are expressed in terms of two switching signals we must employ the divisor shown in Fig. 6.

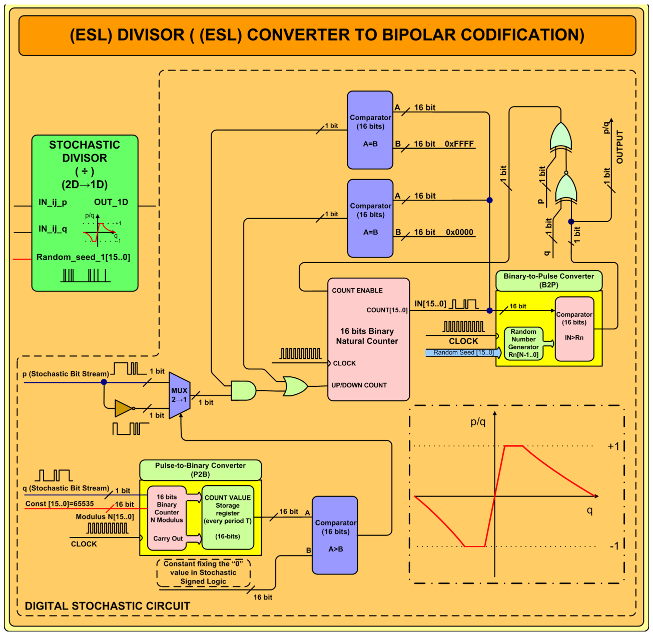

(ESL) DIVISOR ((ESL) CONVERTER TO BIPOLAR CODIFICATION)
STOCHASTIC DIVISOR (÷) (2D→1D)
Binary-to-Pulse Converter (B2P)
16 bits Binary Natural Counter
Pulse-to-Binary Converter (P2B)
DIGITAL STOCHASTIC CIRCUIT
OUTPUT p/q

Fig. 6: (ESL) Stochastic divisor, converts two (ESL) p/q signals in the equivalent bipolar signal "output=p/q"

The proposed divisor is able to estimate the ratio *p/q* of numerator and denominator by using a feedback between the output of an up/down counter and a binary to pulse converter (B2P). The output of the B2P converter (that must be the expected *p/q*) is selected such that its product with signal '*q*' (through the XNOR gate) must be approximately equal to *p*. If this is the case, then the output of the up/down counter is stable, otherwise this output converges to the desired value. If the ratio *p/q* is over *+1* (or under -1) then the output of the divisor saturates to *+1* (or *-1*).

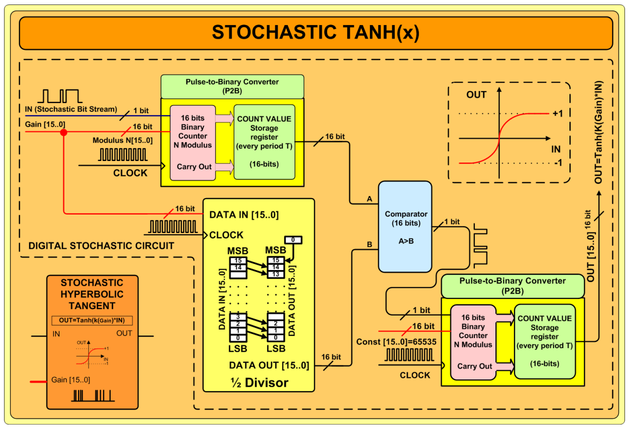

STOCHASTIC TANH(x)
Pulse-to-Binary Converter (P2B)
Comparator (16 bits) A>B
½ Divisor
STOCHASTIC HYPERBOLIC TANGENT
DIGITAL STOCHASTIC CIRCUIT
OUT=Tanh(K(Gain)*IN)

Fig. 7: Diagram of stochastic (bipolar) Tanh(x) digital circuit

The Hyperbolic Tangent function is in fact an approximation to the real function. It is based on the similarity of the cumulative distribution function of a binomial probability distribution with the hyperbolic tangent function. The cumulative distribution function is obtained integrating a single stochastic signal that is itself governed by a binomial distribution. Then the output of an integrator of such a signal is:

$$p = Stochastic\_Input\_Signal$$

$$f(k) = \binom{Gain}{k} \cdot p^{k} \cdot (1-p)^{Gain-k} \ \forall \mathrm{k} = 0,1,\ldots,\mathrm{Gain} \rightarrow$$

$$\text{comparator} \rightarrow \begin{cases} f(k) \leq \dfrac{Gain}{2} \rightarrow out = 0 \rightarrow out^{*} = -1 \\ f(k) > \dfrac{Gain}{2} \rightarrow out = 1 \rightarrow out^{*} = +1 \end{cases} \Rightarrow \quad (4)$$

$$\Rightarrow Output = F(x) = \sum_{u=0}^{x} \binom{n}{u} \cdot f(k)^{u} \cdot (1 - f(k))^{n-u} \quad \forall\, 0 \leq \mathrm{x} \leq \mathrm{n}$$

Where "p" is the input signal switching probability and Gain is fixed by the user. The hyperbolic tangent block (Fig. 7) is composed by a P2B converter, a comparator and an external P2B block. The P2B converter counts the number of high values of the input switching signal IN during a total of cycles equal to the binary number Gain (16 bits register). Finally, a comparator compares the result of such count with Gain/2 in order to convert the output value of the P2B(N) block to a pulsed signal. The comparator output is integrated over 2^{16} clock cycles by the second P2B(N) block. The output result is a function similar to the hyperbolic tangent expression.

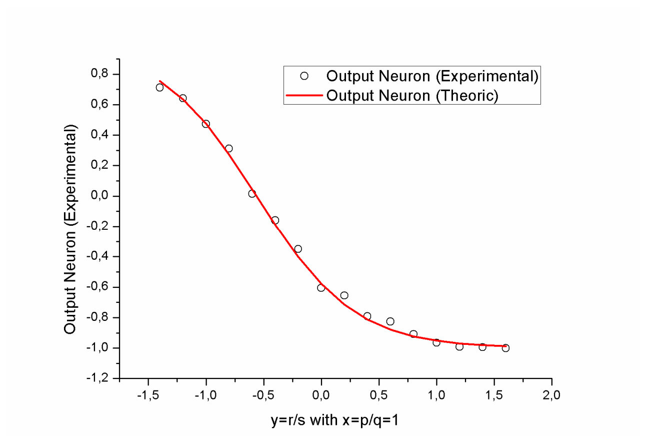


Fig. 8: Tangent-Sigmoid neuron experimental measurements (symbols) vs. expected behavior (solid line)

The gain *"k"* of the hyperbolic tangent *Tanh(kx)* is related to the value of circuit parameter **Gain**. For the case in which **Gain**=41 we have that *k=4.0687*. If we want to select other values of **Gain** the parameter *k* must be recomputed, but in general **Gain** must be an odd number and increases as **Gain** increases.

Therefore, the non-linear part of the Stochastic Neuron (composed by the divisor and the hyperbolic tangent block) is

able to evaluate the result of the weighting sum of the inputs (represented by two switching signals, the numerator and the denominator) through the non-linear hyperbolic tangent activation function. In the estimation of the weights we must take into account that the non-linear function implemented by the proposed block (*Tanh(kx))* is weighted by a *k* value equal to *4.0687* (that normally depends on the value of **Gain** that we equate to *41*).
Once the basic building circuits are stated we can construct Artificial Neural Networks using the proposed schemes. Note that the advantage of ESL with respect traditional stochastic computing elements is that the weights of the networks can be arbitrary high (and not bounded between -1 and +1). Therefore, the solution provided by traditional training algorithms (as those implemented in MATLAB) can be directly applied to the proposed implementation, only adjusting its values by a factor *1/k*; in order to reflect the effect of the hyperbolic tangent block.

To test the proper operation of the Tangent-Sigmoid neuron developed we have proceeded to repeat the same example developed for linear neuron (2 inputs + 1 bias neuron). In this case we vary the input "x_2" in the interval $\{-1.4,+1.6\}$. In Fig. 8 we show the measurements of the output of the Tangent-Sigmoidal neuron with respect variations at the input "x_2". In this figure the results obtained (symbols) fit perfectly with respect to the expected behavior (solid line).

As you can observe, the circuitry to generate the hyperbolic tangent is very simple, requiring very few logical resources. With respect to the generation of the activation function a similar scheme were proposed by Bade et al. [6] for unipolar encoding, and a completely different proposal for bipolar encoding has presented by Brown et al. [2].

The main drawback of the proposed methodology is the same problem present in the traditional stochastic logic, the precision, since switching signals present a fluctuation around the expected value that depends on the integration time used by the P2B converters.

IV. Practical Implementation of Artificial Neural Networks

In previous section we showed the basic neurons used to implement a complex neural network. In this section we present a simple application to demonstrate the feasibility of using the ESL structure in the implementation of complex neural networks. We have implemented a simple classifier based on a neural network that is able to classify between two different categories, represented by [A, B]. For this we have implemented a full interconnected feed-forward network of three layers, an Input layer composed by the two inputs (X, Y), a hidden layer composed by five linear neurons and an output layer composed by one non-linear neuron. The two inputs (X, Y) represent the measurements to recognize and the output neuron classifies the input (with logic *'1'* class A or a logic *'0'* value class B).

Firstly we have generated two training populations using a Monte Carlo based technique. Then we trained the network with the "nntool" utility from MATLAB using the Back-Propagation training technique, and the values of weights and bias obtained are directly implemented in hardware (FPGA), only readjusting by a factor (1/4.0687). The design of the Artificial Neural Network was realized combining VHDL code and Quartus II schematic blocks and the design was implemented in an ALTERA Cyclone II EP2C20F484C7N low cost FPGA.

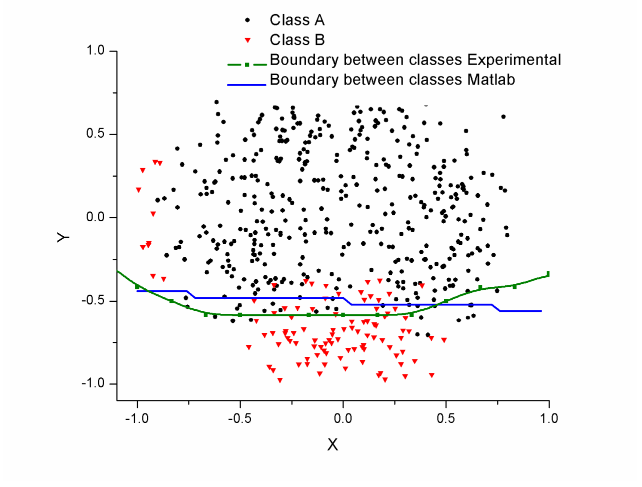


Fig. 9 The implemented network is able to distinguish between two classes (black and red symbols). The expected behavior of the trained network is presented by the blue line while the green line presents the experimental boundary line between classes

As can be appreciated in Fig. 9, the experimental measurements (green line) are very close to the expected behavior (blue line) but are not equal, demonstrating that the network is able to recognize the two different classes (black (A) and red (B) symbols). These results show that the proposed methodology is applicable to the stochastic Neural Network implementation and that the results of the optimum weights provided by standard/classic training algorithms are directly applicable to the network.

V. Conclusions

In this work we presented a new computation methodology based on stochastic logic. The proposed scheme uses two switching signals to define any real number that enable to operate with any real number. The proposed computation method circumvents the problem of traditional stochastic logic of being only defined between 0 and +1 for the unipolar notation (or between -1 and +1 for the bipolar notation). Different blocks are presented that are able to efficiently implement Artificial Neural Networks. In particular, linear and non-linear neurons (using the Tangent-Hyperbolic activation function) can be implemented. With the proposed methodology, complex mathematical expressions can be obtained with low hardware resources, thus allowing the implementation of massive parallel computing. The other advantage of the proposed methodology is that it is fully digital and therefore con be easily implemented in Field-Programmable Gate Arrays (FPGAs).

The shortcoming of the methodology is the error in the conversion and the precision of the implementation since stochastic signals are always subject to an error associated to the random fluctuation of signals.

ACKNOWLEDGMENT

This work was supported in part by the European Development Funds (FEDER) and in part by the Spanish Government under grant TEC2011-23113

REFERENCES

[1] Judith E. Dayhoff, "Neural network architectures: An introduction", Wiley & Sons, 1990

[2] B. Brown and H. Card, "Stochastic Neural Computation I: Computational Elements", IEEE Transactions on Computers, Volume. 50, Issue. 9, pp. 891–905, 2001

[3] J.L. Rosselló, V. Canals, A. Morro "Hardware implementation of stochastic-based Neural Networks", In Proc. International Join Conference on Neural Networks (IJCNN 2010), Barcelona (Spain), 2010

[4] C.L. Janer, J.M. Quero and L.G. Franquelo, "Fully parallel summation in a new stochastic neural network architecture", in Proc. IEEE international Conference of Neural Networks, San Francisco (CA) (U.S.A), pp. 1498-1503, 1993

[5] A. Torralba, F. Colodro, E. Ibanez, L.G. Franquelo, "Two digital circuits for a fully parallel stochastic neural network", IEEE Transactions on Neural Networks, volume 6, issue 5, pp. 1264-1268, 1995

[6] S.L. Bade, B.L. Hutchings, "FPGA-based stochastic neural networks-implementation", On proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa Valley (CA) (U.S.A), pp. 189-198, april 1994

[7] Y. Maeda and Y. Fukuda, "FPGA Implementation of Pulse Density Hopfield Neural Network", In Proc. Int. Joint Conf. on Neural Networks, Florida (U.S.A), 2007

[8] Kondo Y, Sawada, Y, "Functional Abilities of a Stochastic Logic Neural Network", IEEE Trans. on Neural Networks, Volume 3, Issue 3, pp. 434-443, 1992

[9] S. Sato, K. Nemoto, S. Akimoto, M Kinjo and K. Nakajima, "Implementation of a New Neurochip Using Stochastic Logic", IEEE Trans. on Neural Networks, Volume 14, Issue 5, pp. 1122-1127, 2003

[10] B.R. Gaines, "Random pulse machines", IEEE Transactions on Computers, pp. 410-410, 1968

[11] B.R. Gaines, "Stochastic computing systems", Advances in Information Systems Science, Volume. 2, pp.37-172, 1969

[12] W.J. Poppelbaum, C. Afuso and J. Esch, "Stochastic computing elements and systems", in AFIPS FJCC, nº31, 1967

[13] S. Buchholz and N. Bihan, "Polarized signal classification by complex and quaternionic multi-layer perceptrons", International Journal of Neural Systems, Volume 18, Issue 2, pp. 75-85, 2008

[14] W. Zou, Z. Chi and K. Lo, "Improvement of image classification using wavelet coefficients with structured-based neural network", International Journal of Neural Systems, Volume 18, Issue 3, pp. 195-205, 2008

[15] A. Khashman, "Blood cell identification using a simple neural network", International Journal of Neural Systems, volume 18, Issue 5, pp. 453-458, 2008

[16] C. Zanchettin and T. Ludermir, "Hybrid neural systems for pattern recognition in artificial noses", International Journal of Neural Systems, Volume 15, Issue 1-2, pp. 137-149, 2005

[17] F. Rasheed, M. AlShalalfa and R. Alhajj, "Adapting machine learning technique for periodicity detection in biological sequences", International Journal of Neural Systems, Volume 19, Issue 1, pp. 11-24, 2009

[18] D. Theodoridis, Y. Boutalis and M. Christodoulou, "Indirect adaptive control of unknown multi-variable nonlinear systems with parametric and dynamic uncertainties using a new neuro-fuzzy system description", International Journal of Neural Systems, Volume 20, Issue 2, pp. 129-148, 2010

[19] A. Panakkat and H. Adeli, "Neural network models for earthquake magnitude prediction using multiple seismicity indicators", International Journal of Neural Systems, Volume 17, Issue 1, pp. 13-33, 2007

[20] H. S. Park and H. Adeli, "Optimization of space structures by neural dynamics", Neural Networks, Volume 8, Issue 5, pp. 769-781, 1995

[21] H. S. Park and H. Adeli, "Distributed neural dynamics algorithms for optimization of large steel structures", Journal of Structural Engineering, ASCE, Volume 123, Issue 7, pp. 880-888, 1997

[22] C.L. Janer, J.M. Quero, J.G. Ortega and L.G. Franquelo, "Fully parallel Stochastic computation architecture", IEEE Transactions on signal processing, Volume 44, Issue 8, pp. 2110–2117, 1996

[23] V. Canals, A. Morro, J.L. Rosselló, "Stochastic-based pattern-recognition analysis", Pattern Recognition Letters, Volume 31, Issue 15, pp. 2353-2356, 2010

[24] Wolfgang Maass, Christopher M Bishop, "Pulsed neural networks", M.I.T. Press, Cambridge (U.S.A), 1999