Generalized Artificial Neural Networks (GANN)

S. Mehdi Fakhraie, K.C. Smith
Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario, M5S 1A4

Abstract - Based on various styles of published artificial neural networks (ANN), yet inspired by our desire to employ those aspects of neuro-computation theory which are more compatible with current VLSI implementation techniques, we describe a general model for neural networks with localized storage parameters. As a special case covered by the model introduced, we employ a quadratic relation similar to that found in practical MOS devices to implement synapses in ANN. A simulator is developed to account for the activity of synapses and to apply additional constraints for bounded weights. Application of one particular version of quadratic synapses has been examined, and positive results are shown both for the simulation of logical functions of two variables, and the detection of characters on a 3x5 retina. Also a geometrical interpretation is derived, which emphasizes that this approach has significant advantages if utilized in nonlinear pattern-recognition problems regardless of its hardware-implementation convenience.

INTRODUCTION

Neural-network researchers have introduced many new techniques and algorithms at the software and system level. In this development, many new concepts have been promoted [1]. In the whole process, the concepts of training and learning have established their place besides traditional techniques of programming. Machines with the capability of generalization and the potential to reach a reasonable solution when faced with never-seen cases are now familiar. Associative memory and unsupervised data clustering are possible and accepted concepts. As a result, the software and algorithmic research communities have been able to advance and thereby introduce many new concepts. One might ask the question: " Should the hardware-research community expect the appearance of similar dramatic changes and the introduction of new techniques and concepts in their field? "

In the past few years, much effort has been directed towards VLSI implementation of ANN [2]. Many problems have been revealed and some of them are partially solved. Low power consumption [3], and accurate analog computational techniques are under investigation [4]. Noise and offset reduction through the process of adaptation now has been shown to be possible [5]. The capability of the training process to encapsulate knowledge embedded in the input data have already been shown. Moreover, the hardware community has shown that it is also possible to exploit a training process to characterize the processing hardware itself [5]. Moreover, consistently, it has been shown that functionality of a basic MOS device can be increased to correspond to that of a simple neuron with constant weights [6].

In our approach, we try to maintain the most important aspect of neural computational theory, which we view to be that of simple interconnected-adaptable processing elements. As to the issue of how to implement basic blocks and their interconnects, instead of following the conventional approach of compiling software blocks in hardware, we have tried to exploit hardware resources as they naturally exist, especially in the form of MOS transistors embedded in a CMOS technology. From the user's point of view, the outcome may be quite the same as with the conventional approach. However, it is possible that internal processing based upon different internal implementations for interconnects and processing elements can lead to a much more efficient design. In this paper, our early attempts to explore some new hardware-based possibilities are investigated.

GENERALIZED ARTIFICIAL NEURAL NETWORKS (GANN)

The neuron is the basic element of a neural network. It performs a simple processing function. A very large and complex network is constructed by interconnecting these simple repeatable blocks. In a general neural network, each neuron also performs a storage function intended to encapsulate external knowledge in its internal structure.

Overall, each neuron is accompanied with a functional characteristic equation relating its output with the incoming signals and the information embedded in memory function. In this paper we concentrate on those neurons having a localized model of storage and processing of information, especially those currently using a linear synaptic relation. This implies that we accept the abstract model for each neuron shown in Figure 1.

This model is a generalization of the original one proposed by McColloch and Pits and also is in accordance with the concepts introduced in basic texts in the field [7]. In the original model, each synapse has a stored value (W_k, called the synaptic weight) and linearly mixes its input (i_k) with the stored or memorized value and provides the synaptic output (output = $W_k \cdot i_k$). Knowledge is encapsulated in the network through these memorized synaptic values. In this model, we assume only a general synaptic mixing function without using the linear combination convention. The next stage is the neuronal integration level, which we

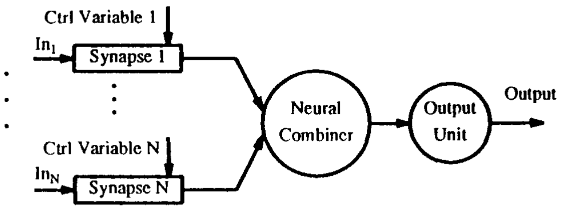

Fig.1. General model of a neuron as discussed in this paper.

suppose is another mixing function with one output but having inputs from all synapses of a neuron. In the McColloch and Pitts model, this stage is a linear summer and the output characteristic function is a hard limiter. Other researchers [1] have proposed various types of saturating or linear output units and we follow this approach.

The original model, called a perceptron model, has attracted a great deal of attention for its capability to classify and/or recognize simple features. In general a perceptron can divide an N-dimensional feature space into two distinct half spaces by placing an N-1 dimensional hyperplane within it. Correspondingly, our proposed generalized nonlinear model is capable of placing an N-1 dimensional hypersurface in an N-dimensional feature space. For example, as will be shown in a later section, our MOS-compatible neurons with quadratic synapses take advantage of hyperspheres to divide the feature space into two parts. This approach not only has certain advantages in the hardware implementation, but also is advantageous in many applications dealing with nonlinearly-separated patterns. For example, one neuron with quadratic synapses can precisely discriminate points inside and outside a circle, while theoretically an infinite number of perceptrons with linear synapses are needed to do the same job. From this generalized point of view, as long as the final goal is achieved, we do not favor linear combiners or summers over nonlinear ones. Rather, if some characteristic is more compatible with implementable sub-blocks in a particular technology, we prefer it in comparison to other, seemingly simpler approaches. This is why in silicon, we favor quadratic synapses in an MOS technology, and, correspondingly, exponential ones in a bipolar process.

The characteristic equation we use for a generalized neuron is:

$$Yout_j = F(G(H(W_{1j},v_1),...,H(W_{Nj},...,v_N)))$$

where W_{ji} are the synaptic controllable parameters which encapsulate information during the training process, and v_i are inputs to that neuron. Inputs and synaptic information are mixed by the nonlinear function H(,) for each i. Then, different results for all i are combined by the G( ) function. Finally, the output of each neuron is defined using the nonlinear F( ) function.

Having considered synaptic memories and combining functions and neuronal mixing and output functions, the most important job which remains is to identify a plausible training algorithm by which to adapt the synaptic weights. With some pre-defined or arbitrarily selected error or performance measure, one should follow an algorithm to find a set of parameters which make the performance measure better than a given threshold. Another flexibility available in general here, and exploited by several authors, is the selection of error or performance-evaluation function [8]. Since training of our parametric network is a nonlinear optimization problems, we have been able to take advantage of the rich literature in the field [9].

Possible Variations of GANN

The concept of implementing input-output transformations using the idea of generalized superposition of time-/space-/frequency-localized basis functions is employed in signal-processing applications. These ideas can be re-structured in a multi-layer network form. For example, in a related research work [10], we have employed a 3-layer network structure to find a modal-analysis solution for an electromagnetic problem with sampled-noisy-measured data instead of conventional analytic stimulus functions. The three input units represent a point in input space. Each neuron in the hidden layer has a three-dimensional sinusoid as its characteristic function. The output unit is a linear unit adding the outputs of hidden-layer sinusoids with some connection weights. Sampled input-output pairs are applied to the network and training or adaptation schemes similar to error back propagation is followed to adjust the connection weights. This resembles a Fourier-analysis-based modal expansion. However, it can take advantage of the merits of a neural network solution: among them being noise suppression and the capability of employing noise-corrupted sampled input data [10]. This approach is similar to compiling an old program to a new language in order to utilize the capabilities of a new hardware/software platform.

Other attempts at generalization can be found in the literature: A good example is the fuzzy neuron which can be regarded as a special case covered by the present general model [11]. In the hardware community, Lont et al [12] have used non-perfect linear units with an emphasis on a new look at theory rather than simply considering their synaptic unit as defective.

In this paper, we will introduce the application of MOS-compatible quadratic synapses to ANN's. Other variations of abstract neurons with quadratic characteristic functions had been proposed [13]. However, our attempt is the first we have found in the literature to establish a link between generalized definitions (including quadratic ones) and MOS-compatible hardware implementations.

NONLINEAR MOS-COMPATIBLE QUADRATIC SYNAPSES

In hardware implementations, silicon MOS technology, with its widespread application and high integration level, is among the most promising candidates. Correspondingly, it has been employed in many hardware-based implementations of ANN [2]. A simple MOS transistor is the basic applicable block available in the process. The characteristic of such a building block, when employed in a circuit, can be approximated by a quadratic function over a wide operating range. Also in a normally smaller part of the operating range, a nominally linear dependence, which has been used by several researchers [14], can be found. Here, we explore the possibility of using quadratic synapses in a feed-forward perceptron configuration. This approach is inspired by the belief that more reliable devices with wider operating range and higher speeds can be employed if it is possible to use the naturally-available quadratic characteristic equation for the most widely-used block of an ANN, namely the synapse.

A quadratic synapse is implemented by the equation: $i_{out} = (v_{in} - W)^2$, where "W" is the synaptic weight, or memorized information. In a linear synapse, the resulting stimulation is inhibitory or excitatory depending on the sign of the output. Obviously, the quadratic model in the form above supports only excitatory stimulation. In order to provide either inhibition or excitation directed by the magnitude of only one control parameter (W), we have adopted the following synaptic relation: $i_{out} = (v_{in} - W)^2 - 1$.

In this equation, "W" can take either of negative or positive values. Depending on its magnitude in comparison to the input values, i_{out} becomes positive (excitatory) or negative (inhibitory). In consideration of its accompanying hardware implementation, we call this implementation the Current-Source-Inhibited (CSI) quadratic synapse. Figure 2 depicts the synaptic characteristic relation for such synapses. For a MOS implementation of a quadratic synapse, see [15]. In brief, "W" is realized by an externally controllable threshold voltage of an MOS device; "v_{in}" is the gate-source voltage of the MOS transistor ;and i_{out} is best approximated by its drain current with operating in current saturation. As stated in [5, 12], the hardware should be used in the training process, at least in its forward pass, in order to account for circuit imperfections. Under this condition, just an approximate characteristic suffices, and it is not necessary to use devices which exactly follow the quadratic theoretical relation.

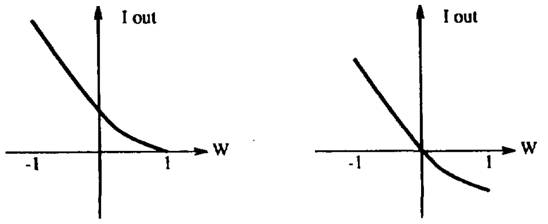

Fig.2. The above graphs show the input/output characteristics of a quadratic (left) and a Current-source-Inhibited quadratic (right) synapse for a high input value equal to 1. "W" is assumed to vary between -1 and 1.

Networks Composed of Quadratic Synapses

Figure 3 shows a 3-layer network built from neurons with quadratic synapses. Here, the neural-integration level, is implemented by the customary linear-addition function. For output unit, the use of $y_1 = \frac{1}{1+e^{-x}}$ has been investigated.

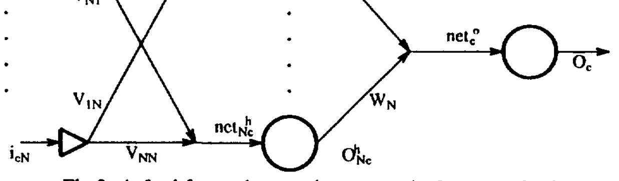

Fig.3. A feed-forward network composed of neurons having quadratic synapses. Output of each synapse is equal to : $[(\text{input} - \text{weight})^2 - 1]$.

Each neuron has a number of quadratic synapses having a " $i_{out} = (v_{in} - W)^2 - 1$ characteristic," connecting it with inputs or neurons from the previous layer. A bias input constantly equal to the active-high level (notated here as equal to one) is also assumed.

In this paper, we aim at presenting the results of our feasibility studies. However, in order to maintain a close connection with hardware implementations, we have applied several other constraints not deemed necessary in usual software simulations. For example, we have assumed that v_{in} should be greater than "W" in order to establish an active synaptic connection between a neuron and each of its inputs. Basically this converts the training problem from an unconstrained optimization problem (used in the back-propagation error scheme) to one of constrained optimization. As an error function, we use the mean-squared error. This means that our training challenge is to solve the following constrained optimization problem, where E( ) is the error function:

$$\text{Minimize:} \quad E(\overline{W}, \overline{v_{in}})$$

Subject to: $v_{in_i} - W_i \geq 0, i = 1,\dots,P$, where P is the number of synapses.

These additional limitations may look inconvenient and restrictive. However, the major point is that no unconstrained optimization scheme can actually be employed in a physical implementation.

TRAINING EQUATIONS

In order to adjust the adaptable parameters of the network, we use a gradient-descent update procedure. Our training algorithm is a modified version of the back-propagation algorithm conventionally used for linear synapses [7]. For simplicity, we consider the training equations for a 3-layer network with one output unit. Conventions used are similar to those of Figure 3, and are introduced here in detail:

E_{total} :	E_{total} is equal to $\sum_{1}^{Cases} E_c$; "Cases" is the number of points in the training set.
E_c :	$E_c = \frac{1}{2}(O_c - O_{c,desired})^2$, is the error for each training case.
O_c :	Real output of the network for training case c.
$O_{c,desired}$:	Desired output of the network for training case c.
W_j :	A weight connecting the jth hidden neuron to the output neuron.
V_{ji} :	A weight connecting the ith input to the jth neuron in the hidden layer.
net^o_c :	Total input received by the output neuron.
O^h_{jc} :	Total output of the jth hidden neuron.
net^h_{jc} :	Total input received by the jth hidden neuron.
i_{ic} :	Input applied to the ith input unit in case "c."

A. Calculation of the Gradient for the Synapses of the Output Layer

$$\frac{\partial E_{total}}{\partial W_j} = \sum_{c=1}^{Cases} \frac{\partial E_c}{\partial W_j}$$

$$\frac{\partial E_c}{\partial W_j} = \frac{\partial E_c}{\partial O_c}\cdot\frac{\partial O_c}{\partial net^o_c}\cdot\frac{\partial net^o_c}{\partial W_j}$$

We call $\delta_c = \frac{\partial E_c}{\partial O_c}\cdot\frac{\partial O_c}{\partial net^o_c} = O_c\cdot(1 - O_c)\cdot(O_C - O_{c,desired})$, and

$$\frac{\partial E_c}{\partial W_j} = (\delta_c)\cdot(-2)\cdot(O^h_{jc} - W_j)$$

B. Calculation of the Gradients for the Synapses in the Hidden Layer

$$\frac{\partial E_c}{\partial V_{ji}} = \frac{\partial E_c}{\partial O_c}\cdot\frac{\partial O_c}{\partial net^o_c}\cdot\frac{\partial net^o_c}{\partial O^h_{jc}}\cdot\frac{\partial O^h_{jc}}{\partial net^h_{jc}}\cdot\frac{\partial net^h_{jc}}{\partial V_{ji}}$$

As seen, the first two terms are the δ_c calculated above. Also, we introduce $\delta^h_{jic} = \delta_c\cdot\frac{\partial net^o_c}{\partial O^h_{jc}}\cdot\frac{\partial O^h_{jc}}{\partial net^h_{jc}}$, which can be used in the training of networks with deeper hidden layers. Therefore

$$\delta_{jic} = \delta_c\cdot 2\cdot(O^h_{jc} - W_j)\cdot O^h_{jc}(1 - O^h_{jc})$$

$$\frac{\partial E_c}{\partial V_{ji}} = \delta_{jic}\cdot(-2)\cdot(i_{ic} - V_{ji})$$

C. Updates

In an stochastic gradient approach [1], an incremental update for V_{ji} should be in proportion to $-\frac{\partial E_c}{\partial V_{ji}}$. It means that $\Delta_c V_{ji} = -\eta_{ji}\cdot\frac{\partial E_c}{\partial V_{ji}}$. The same concept applies to W_j: $\Delta_c W_j = -\eta_j\cdot\frac{\partial E_c}{\partial W_j}$.

η_{ji} and η_j are training coefficients. Normally they are set to a small number for a differential movement. A proper algorithm to adjust and potentially change them can help to achieve faster convergence and to escape from undesirable situations.

In batch training, each update takes place after one pass through all training cases:

$$\Delta V_{ji} = \sum_{c=1}^{Cases} \Delta_c V_{ji} \; ; \qquad \Delta W_j = \sum_{c=1}^{Cases} \Delta_c W_j$$

$$V^{new}_{ji} = V^{old}_{ji} + \Delta V_{ji} \; ; \qquad W^{new}_j = W^{old}_j + \Delta W_j$$

In any implementation, including ours, we are dealing with bounded weights. Theoretically, Stinchcombe et al [16] have shown that networks with bounded weights can perform universal function approximation with an arbitrarily small error; however, they require a larger number of neurons. In our case, as shown in simulations, utilization of a larger number of hidden units improves the performance of bounded-weight networks.

SIMULATIONS AND VERIFICATION OF THE APPROACH

A. Simulator Developed

Since our ultimate goal is to provide improvements in hardware implementations, we need to investigate many different types of synaptic relations, neural integration, and output activity functions. Also we need a constrained-optimization algorithm for adjusting the synaptic weights. The possibility of having different objective functions and constraints is desirable as well. In general, then, we require additional flexibility to examine and explore the effects of many different parameters involved in a possible hardware implementation.

For all of the above reasons, we could not use one of the existing general-purpose neural-net simulators. Therefore we have developed our own software simulator to investigate different variations of generalized ANN including those with quadratic synapses.

We have used our simulator to solve several classic problems using different variations of networks with quadratic synapses. Here, we discuss our attempts with three-layer networks using CSI quadratic synapses:

B. Logic Functions of Two Variables

In this part, we describe how a two-input one-output feed-forward network is trained to implement an arbitrarily selected logic function of two variables. In order to fairly evaluate the performance of quadratic synapses, we first describe the performance of a standard back-propagation three-layer network with linear synapses. It is well known that the XOR function is among the most difficult problems to be solved by perceptrons, a problem which is impossible to solve with perceptrons having only two layers [7]. Therefore we use the XOR problem to compare two types of networks:

B.1. Performance of Perceptrons with Linear Synapses

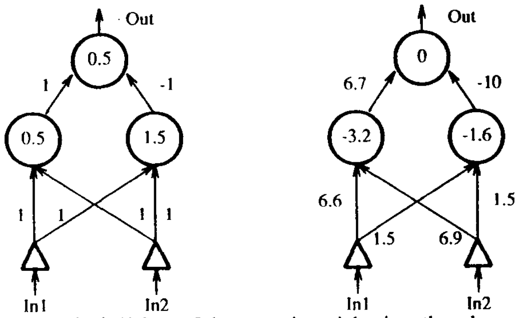

Fig.4. Values of the synaptic weights in a three-layer perceptron with linear synapses. Left: A theoretical solution. Right: A practical solution obtained by using a back-propagation training algorithm with output target values at 0.2 and 0.8 .

Figure 4 illustrates the values of the synaptic weights after a three-layer perceptron with linear weights is trained to solve the XOR problem. Two networks are shown: the first one is a perceptron with a hard-limiter output function. In this network the weights are no larger than 1.5 . The other network utilizes the $\frac{1}{1+e^{-x}}$ output function and has been trained with back propagation starting from a random initial point. The above output function will take output values of one or zero only when its argument is infinitely large. For target outputs at 0.2 and 0.8, the weights shown in Fig. 4 are obtained. As seen, weight values 10 times larger than the maximum level of the i/o signals are obtained. Therefore, in a standard back-propagation scheme with bounded weights, there is a large possibility of getting stuck in non-optimum points for some of the random initial points.

B.2. Performance of Perceptrons with CSI Quadratic Synapses

We have trained a network with i/o levels at zero and one, where the synaptic values in the quadratic relations vary between -1 and 1 . The network has two input units, one hidden layer with all units fully connected to inputs using quadratic synapses, and one output unit. The synaptic relation is:

$$i_{out} = \begin{cases} (v_{in} - W)^2 - 1 & \text{if } v_{in} \geq W \\ 0 & \text{if } v_{in} < W \end{cases}$$

where $0 \leq v_{in} \leq 1$ and $-1 \leq W \leq 1$. The restrictions applied to this network are two-fold: one is the bounds on W's, the other is the activity condition ($v_{in} > W$). Weights are randomly initialized and the network is trained by employing the training algorithm previously discussed. Table 1 reflects the results obtained with different numbers of hidden units. The stated high and low output levels are averages obtained in numerous simulations with different starting points and final target values at the outputs. Normally after these levels are reached, most of the weights begin to limit at one of the -1 or 1 extremes if training is continued.

Table 1. Simulation results for logical functions of two input variables obtained using a three-layer network with CSI quadratic synapses. Low and High output levels are those obtained after training is completed.

Test Problem	No. of Hidden Units	High Output Level	Low Output Level
XOR	2	57%	36%
XNOR	2	72%	38%
XOR	3	62%	29%
XNOR	3	75%	30%
XOR	4	63%	29%
XNOR	4	79%	23%
XOR	5	69%	28%
XNOR	5	87%	21%
XOR	6	70%	25%
XNOR	6	89%	17%

C. A Simple Character-Recognition Application

In another test problem, We have trained a network with a 3x5 input retina to recognize images of input digits. The network has been tested with different numbers of hidden units, and, for each digit, one output unit is introduced. All networks with more than one hidden unit can solve the problem. In each case, the worst-case low and high output levels (i.e. the highest low and the lowest high) are reflected in Table 2.

Table 2. Worst-case low and high output levels in a simple character-recognition problem.

Network	Highest Low	Lowest High
2 Hidden Units	41%	73%
3 hidden Units	29%	81%
6 Hidden Units	8%	94%

D. Geometrical Interpretation

Suppose we have a neuron composed of N quadratic synapses, M of which are active for a particular input vector. Also for clarity, assume that the neuron has a hard-limiter output unit. Then, the total input of the neuron is

$$\text{net}_{\text{input}} = \sum_{i=1}^{M}\left[(V_i - W_i)^2 - 1\right] = \sum_{i=1}^{M}(V_i - W_i)^2 - M .$$

Therefore, if $\sum_{i=1}^{M}(V_i - W_i)^2 > M$, $\text{Output}_{\text{neuron}} = 1$, otherwise $\text{Output}_{\text{neuron}} = 0$.

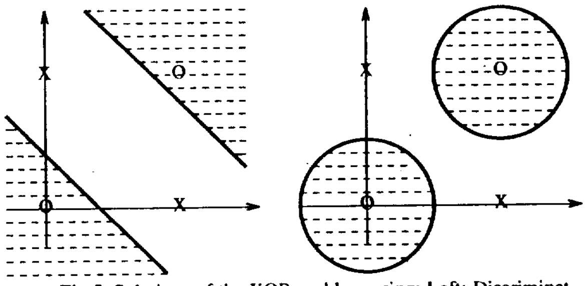

Fig.5. Solutions of the XOR problem using; Left: Discriminating lines. Right: Discriminating circles.

However, $\sum_{i=1}^{M}(V_i - W_i)^2 = M$ is the equation of a hypersphere with radius $\sqrt{M}$. It means that if the point $(V_1, \cdots, V_M)$ is inside the hypersphere, the output is high, otherwise the output is low. Each neuron accepts a sub-vector of its input vector with components greater than adjustable thresholds. Then, it determines whether that sub-vector is inside a discriminating hypersphere or not. Figure 5(right) shows how discriminating circles can be used to solve the XOR problem. Figure 5(left) shows a conventional solution for XOR using discriminating hyperplanes (lines in two-dimensional space). Although MOS-compatibility was our original motivation in utilizing quadratic synapses, these elements with their discriminating spheres can be of greater help than linear discriminating functions in many nonlinear pattern-recognition problems.

CONCLUSION

In this paper, the general idea of parallel-interconnected simple processors has been used to design MOS-compatible quadratic synapses. One variation of quadratic synapses has been utilized in networks of artificial neurons to perform simple character-recognition and logical-function tasks. A gradient-descent training algorithm for networks employing quadratic synapses has been developed. Also, a geometrical interpretation and comparison to conventional perceptrons are illustrated. In future work, we will show the performance of other MOS-compatible quadratic synapses and will demonstrate the hardware basis of our proposal.

REFERENCES

1. J. Hertz, A. Krogh, and R.G. Palmer, *Introduction to the Theory of Neural Computation*. Redwood City, CA: Addison-Wesley, 1991.
2. A.F. Murray, " Silicon implementations of neural networks, " *IEE Proceedings-F*, vol. 138, No. 1, pp. 3-12, February 1991.
3. C.A. Mead, *Analog VLSI and Neural Systems*. Reading, MA: Addison-Wesley, 1989.
4. J. Choi, and B.J. Sheu, "VLSI design of compact and high-precision analog neural network processors, " *Int. Joint Conf. Neural Networks*, Baltimore, Maryland, June 7-11, 1992, pp. II-637-642.
5. S. Satyanarayana, Y. P. Tsividis, and H. P. Graf, " A reconfigurable VLSI neural network, " *IEEE J. of Solid-State Circuits*, 27(1) , pp. 67-81, Jan. 1992.
6. T. Shibata, and T. Ohmi, " A functional MOS transistor featuring gate-level weighted sum and threshold operations, " *IEEE Trans. on Electron Devices*, 39(6), pp. 1444-1455, June 1992.
7. D.E. Rumelhart, J.L. McClelland, and PDP Research Group, *Parallel Distributed Processing*. vol. 1, Cambridge, MA: MIT Press, 1986.
8. J.A. Joines, and M.W. White, "Improved generalization using robust cost functions, " *Int. Joint Conf. Neural Networks*, Baltimore, Maryland, June 7-11, 1992, pp. III-911-18.
9. R. Fletcher, *Practical Methods of Optimization*. John Wiley & Sons, 1987.
10. S.M. Fakhraie, A. Konrad, and K.C. Smith, "Neuro-computation techniques in sampled-data electromagnetic field problems," submitted to COMPUMAG'93, Conf. Computation of Electromagnetic Fields, Miami, Florida, Oct. 31-Nov. 4, 1993.
11. T. Yamakawa, and S. Tomoda, "A fuzzy neuron and its application to pattern recognition, " *Proc. Third IFSA Congress*, Seattle, Washington, Aug. 6-11, 1989, pp. 943-8.
12. J.B. Lont, and W. Guggenbuhl, "Analog CMOS implementation of a multilayer perceptron with nonlinear synapses," *IEEE Trans. Neural Networks*, 3(3), pp. 457-465, May 1992.
13. G.S. Lim, M. Alder, and P. Hadingham, "Adaptive quadratic neural nets," *Int. Joint Conf. Neural Networks*, Singapore, Nov. 18-21, 1991, pp. 1943-1948.
14. H.P. Graf, L.D. Jackel, and W.E. Hubbard, "VLSI implementation of a neural network model," *IEEE Computer*, 21(3), March 1988.
15. S.M. Fakhraie, and K.C. Smith, "Synapse-MOS (SYMOS) transistors: Intelligent MOS transistors with learning thresholds," submitted to *Canadian Conf. VLSI*, Banff, Alberta, Canada, Nov. 14-16, 1993.
16. M. Stinchcomb, and H. White, "Approximating and learning unknown mappings using multilayer feed-forward networks with bounded weights," *Int. Joint Conf. Neural Networks*, San Diego, CA, June 1990, pp. III-7-16.