On-Chip Training by Software for Analog Neural Networks Using ANNSyS*

İsmet Bayraktaroğlu
Department of Computer Science and Engineering
University of California San Diego, 9500 Gilman Drive
La Jolla, California 92093-0114
ibayrakt@cs.ucsd.edu

Arif Selçuk Öğrenci, Günhan Dündar, Sina Balkır
Department of Electrical and Electronic Engineering

Ethem Alpaydın
Department of Computer Engineering
Boğaziçi University
80815 Bebek, İstanbul, Turkey
{ogrenci,dundar,balkir,alpaydin}@boun.edu.tr

Abstract - **In this study, a circuit level simulator for analog neural networks based on partitioning techniques has been developed. Behaviour of the analog circuitry is modeled according to its SPICE simulations and those models are used in the initial training of the analog neural networks prior to the fine tuning stage where the simulator has been combined with the Madaline Rule III for hardware training of analog neural networks by software and thus avoiding the effects of circuit nonidealities on neural networks. The software has been tested on many examples and has been shown to be extremely successful.**

1 Introduction

Neural networks have gained popularity in the last few years due to their success in many diverse applications. Neural networks are used when there is no algorithmic solution to a problem or a problem is too complicated to be solved by known algorithms [1]. Also, they can be used when the definition of the problem does not exist, but samples of inputs and corresponding outputs are available. However, many applications require real time or very fast operation. This is possible only with dedicated neural network hardware. Due to the inherently parallel nature of neural networks, digital realizations have not been feasible as digital multipliers require larger area besides being slower. For instance, parallel digital multipliers of 8x8 input word lengths have transistor counts on the order of at least several

*This research is supported by TÜBİTAK under grant number EEEAG-183.

thousand [2]. Analog multipliers of comparable precision use less than 20 transistors. The speed of an analog multiplier is limited only by its bandwidth and can go up to the GHz range. When one looks at neurons, a similar picture can be seen. Again, an adder and the nonlinearity can be realized by less than 20 transistors, whereas the same operations require thousands of transistors in the digital domain [3]-[6]. Analog realizations, on the other hand, have been plagued by non-idealities in circuit behavior. Although many studies have been done on this subject, none of them have yielded a complete solution to this problem; they have rather been content with modeling the effects of circuit nonidealities - like nonlinear synapses [7, 8], neurons that deviate from ideal functions, or errors and limitations in storing weights [9]- bring about. It has been shown in [7] and [8] that multiplier nonlinearity can be a very severe problem even for nonlinearity factors of less than 10% for many applications. In [8], the effects of some non-idealities have been studied through circuit simulation with SPICE and the importance of circuit level simulation in analog neural network design has been demonstrated. Although SPICE is the standard tool for circuit simulation, it is not specially tailored for simulating neural networks. Neural networks consist of the interconnection of many identical blocks so that by partitioning the network during simulation it is expected that the simulation speed increases tremendously.

Different approaches are used to obtain the weights of an analog neural network which also dictate the implementation style and the architecture of the network [1]. These approaches are the following: Non-learning networks, neural networks in analog hardware implementation with externally adjustable weights, and neural networks with on-chip learning.

One commercial implementation of analog neural networks is the ETANN 80170NX chip [4]. This chip has been plagued by limited resolution in storing the synapse weights in that the long time resolution of the weights is not more than five bits. Implementing Madaline Rule III [10] has been suggested for the ETANN chip; however, this requires a host computer and excessive external hardware besides many timing problems which limit the performance of training. Problems like these have prevented the success of this chip in the market so that commercial applications using this chip and similar ones have been very few.

In this work, we present an Analog Neural Network Synthesis System (ANNSyS) based on a circuit simulator for neural networks (ANNSiS) which performs on-chip training on the software by using real life models of the actual synapse and neuron circuitry. The circuit simulator makes use of the fact that neural networks with multilayer perceptron architecture consist of many decoupled blocks. The whole circuit can be partitioned into blocks during simulation and each partition can be simulated separately with the output of one block being the input of the next one. Although simulators employing partitioning techniques have appeared in the literature previously, their application to neural networks have been presented for the first time to the best of our knowledge.

We have incorporated our simulator into the Madaline Rule III which can be used for neural network training when exact models for synapses and neurons are not known. On-chip training is a method to circumvent circuit-level non-idealities in analog neural networks. Although training combined with circuit simulation brings in extra computational load, this can be circumvented by good modeling of the neuron and synapses by which the iterations required by our software will be minimized.

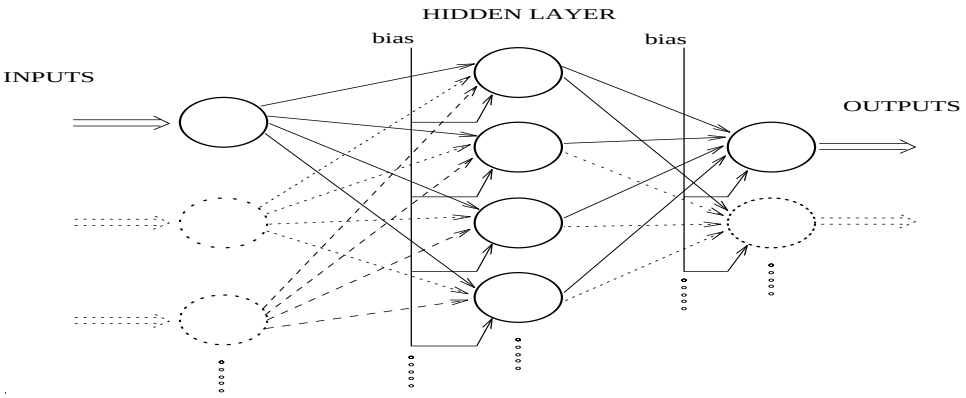

Figure 1: The general structure of the multilayer perceptron.

2 ANNSiS: Analog Neural Network Simulation System

Most neural networks used for pattern recognition or function approximation applications are of the multi-layer perceptron structure. Determination of the weights is performed prior to operation, at the learning phase, which in general, requires the adaptation of weights iteratively to reduce the error (distance between the actual output and the desired output of the network) to zero. Different algorithms can be applied during the learning phase. The most commonly used and well known algorithm is the backpropagation algorithm [10]. The general structure of the so called multilayer perceptron with a single hidden layer is given in Figure 1. Now, we will review the mathematics of the multilayer perceptron where, for simplicity of notation, we limit ourselves to the scalar input-output case. (Solid drawings in Figure 1.) Extension to multidimensional inputs and outputs is straightforward. The output of a multilayer perceptron is a weighted sum of the outputs of a number (h) of hidden units, H_i, filtered through a nonlinear function $\phi(x)$:

$$y(x) = \phi\left(\sum_{i=1}^{h} T_i H_i(x) + T_0\right) \tag{1}$$

where y is the output, x is the input, H_i are the hidden units and T_i are their weights. T_0 is the bias weight. Each hidden unit is another similar sum:

$$H_i(x) = \phi\left(W_i x + W_{i0}\right). \tag{2}$$

Most frequently, the nonlinearity is the sigmoid function:

$$\phi(a) = \frac{1}{1 + \exp(-a)}. \tag{3}$$

Given a training set $\mathcal{X} = \{x^p, r^p\}_p$, where r^p is the desired output corresponding to the input x^p, the sum of squared errors is:

$$E(W,T) = \sum_p E^p = \sum_p [r^p - y(x^p)]^2. \tag{4}$$

In the backpropagation algorithm, parameters W, T are updated to minimize E using gradient-descent method.

Feedforward multilayer neural networks are regular structures where every neuron is connected to every other neuron in the previous layer through synapses. Therefore, they yield themselves easily to partitioning and automatic netlist generation.

The most commonly used tool for circuit simulation is SPICE. However, the size of a neural network circuit for a practical example is very large to be simulated with SPICE. The simulation time of SPICE for neural network circuits increases almost quadratically with the circuit size. Besides, when the circuit size is increased beyond a limit, SPICE starts to have difficulties in simulating the network. This problem can be solved by using partitioning techniques. For DC analysis, if the layers are completely decoupled, by which we mean that the outputs of the neurons are not loaded by the inputs of the synapses of the next layer, the circuit can be partitioned into decoupled blocks which can be simulated separately starting from the input layer. Most of the neural network implementations use CMOS technology. Considering this technology, the assumption of being completely decoupled holds and allows us to partition the network.

ANNSiS is initiated by simulating all partitions in the first layer and finding the outputs. The output values of the first layer are then applied to the next layer as independent voltage sources being input to the synapses of the neurons in that layer. This way, the input is propagated to the output.

We first created different sized analog neural network structures with the building blocks described in Section 3 and simulated them without partitioning by SPICE2G6 and ANNSiS. Next, we simulated them by partitioning into blocks which consist of a neuron and all synapses connected to that neuron. Simulation results of SPICE2G6 and ANNSiS are compared for accuracy and found out to be exactly matching.

When the analog neural network circuit was partitioned, the simulation time decreased remarkably. As seen in Figures 2 and 3, the simulation time increases almost linearly for the partitioned case and almost quadratically for the non-partitioned case. Remarkable decreases in simulation time show the effectiveness of partitioning. Thus, it is possible to simulate large neural network circuits in a faster manner and without any convergence problems.

3 Neural Network Circuitry

Figure 4 shows the circuit used as a synapse. This circuit is a modified version of the well known Gilbert multiplier [11, 12]. The inputs are in the form of voltage differences and are denoted by the couples $x_1 - x_2$ and $y_1 - y_2$. The output of the original Gilbert multiplier is a current difference and this difference is converted to a single ended current (Z) through current mirrors. This improves the linearity of the multiplier as well as providing easy interfacing to the following circuitry. The output current characteristics of the synapse circuit is shown in Figure 6.b. where the current is in microamperes.

Figure 5 shows an OPAMP that can be used to sum the outputs of all the synapses connected to the neuron and convert the current sum into voltage. This OPAMP consists of two stages. The first stage is a differential amplifier whose differential current output is mirrored into the next stage and converted to a single ended output through circuitry very

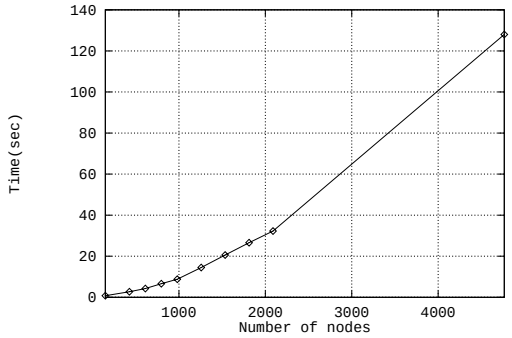

Figure 2: Simulation time for different MLP structures (without partitioning).

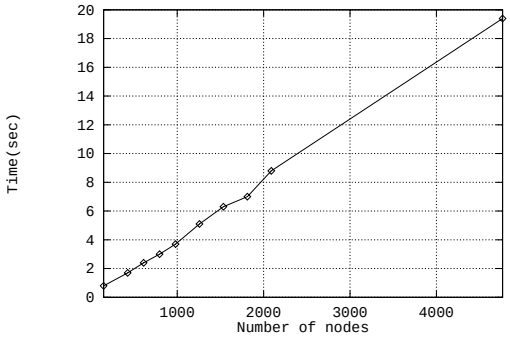

Figure 3: Simulation time for different MLP structures (with partitioning).

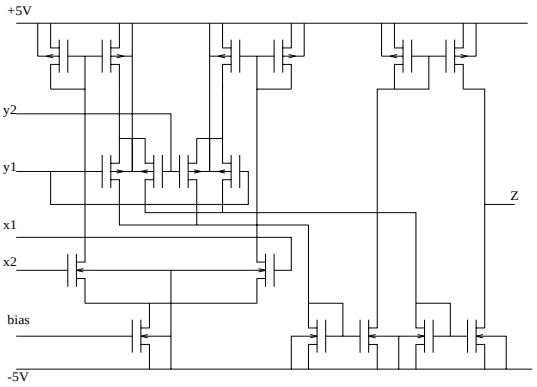

Figure 4: Four-quadrant Gilbert multiplier.

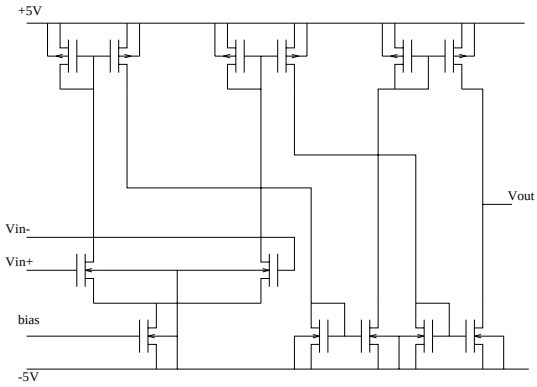

Figure 5: General purpose OPAMP

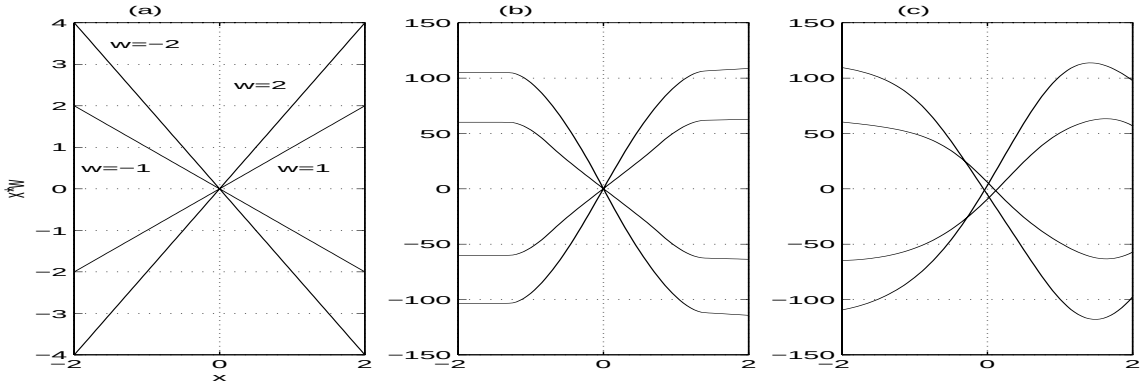

Figure 6: (a) Ideal multiplier (b) Multiplier data provided by SPICE simulation (c) Approximation to (b) used in training.

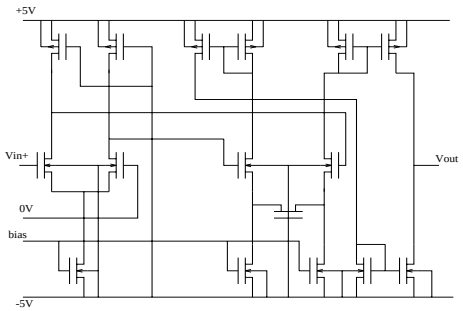

Figure 7: Sigmoid generator.

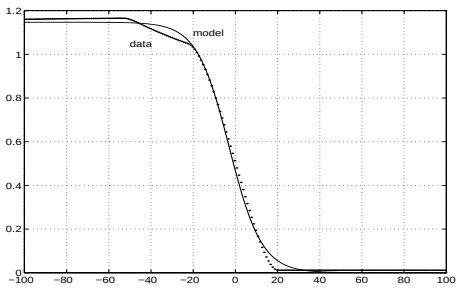

Figure 8: Nonideal sigmoid implemented.

similar to the synapse circuit above. A sigmoid generator introduced in [13] is used after the OPAMP to generate the activation function for the neuron. This generator is depicted in Figure 7, and the characteristics are plotted in Figure 8.

Layouts of the analog neural network circuitry described above are designed in a full-custom manner based on Mietec's 2.0μm technology. The netlists are extracted from the layouts and the simulations for characterization purposes are carried out using HSpice. It is evident from those characteristics that the actual circuits exhibit nonideal behavior so that an adaptation in the training algorithm is required. The nonideality is best visible in the synapse characteristics where the output current curves deviates highly from linearity for large values of inputs and/or weights. Denoting the synapse function (multiplication of the input and the weight:$x \cdot w$) as $\mu(w, x)$, we can rewrite Eqs. (1) and (2):

$$\begin{aligned} y(x) &= \sum_{i=1}^{h} \mu(T_i, H_i(x)) + \mu(T_0, 1) \\ H_i(x) &= \phi\left(\mu(W_i, x) + \mu(W_{i0}, 1)\right). \end{aligned} \tag{5}$$

The equations to update the weights using gradient descent are as follows:

$$\begin{aligned} \Delta T_i &= \eta(r^p - y^p)\frac{\partial \mu(T_i, H_i(x^p))}{\partial T_i} \\ \Delta T_0 &= \eta(r^p - y^p)\frac{\partial \mu(T_i, 1)}{\partial T_i} \\ \Delta W_i &= \eta(r^p - y^p)\frac{\partial \mu(T_i, H_i(x^p))}{\partial H_i(x^p)}\frac{d\phi(x^p)}{dx^p}\frac{\partial \mu(W_i, x^p)}{\partial W_i} \\ \Delta W_{i0} &= \eta(r^p - y^p)\frac{\partial \mu(T_i, H_i(x^p))}{\partial H_i(x^p)}\frac{d\phi(x^p)}{dx^p}\frac{\partial \mu(W_{i0}, 1)}{\partial W_{i0}}. \end{aligned} \tag{6}$$

So the learning method can be implemented using any multiplier if $\mu(w, x)$, $\partial\mu/\partial x$, $\partial\mu/\partial w$, ϕ, and ϕ' are given. However, describing the behavior of the synapses and neurons analytically is not a simple task as will be shown in the next section.

4 Training

There are three different approaches to the hardware implementation of neural networks as explained in the introduction. The non-learning implementation requires minimal chip size, whereas, due to the mismatches between the models and real circuitry, its performance is the worst one, and the on-chip learning implementation gives the best performance. However, if the model of real circuitry perfectly matches the real neural network, the non-learning implementation will be equivalent to on-chip training. An almost perfectly matching model can be the SPICE model of the analog network. Having developed a tool for simulating analog neural networks with SPICE models, Madaline Rule III [10] was chosen to fine tune the weights obtained from the backpropagation algorithm which uses models for the neurons and synapses.

The approximate weights are calculated on a computer with backpropagation algorithm as modified according to Eqs. (5) and (6). One should also modify the learning rule so as to favor small weight values. This is done by the technique named *weight decay*. In weight decay, the error function Eq. (4) is modified to be:

$$E(W,T) = \sum_{p}[r^p - y(x^p)]^2 + \lambda \sum_{i} w_i^2 \tag{7}$$

where w_i are the weights in the network (including both W and T values). The first term is the usual sum of squared errors. The second term is the *penalty* term that penalizes weights that have large magnitude. Performing gradient-descent on this, we get:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w} - \lambda w_i \tag{8}$$

Thus at each iteration, there is an effect of pulling a weight towards zero. So a weight decays towards zero unless pushed away to decrease the sum of squared errors. From a Bayesian perspective, weight decay corresponds to assuming that weights, w_i are sampled from a Gaussian distribution with zero mean and variance $1/\lambda$. Minimizing Eq. (7) is the maximum likelihood solution.

Using weight decay and a suitable choice of λ, w values are found that minimize error and also stay in the linear region of the multiplier (approximately between $-2V$ and $2V$). These weights are then downloaded to the SPICE representation of the network to start Madaline Rule III. Madaline Rule III is an algorithm that is used when the backpropagation algorithm cannot be applied due to the impossibility of finding the derivatives of error with respect to the weights. Instead of using the derivatives, a small amount of disturbance is added to the summation of the synapse outputs and the difference in the error is measured. Dividing the error difference by the disturbance, we find the derivative of the error with respect to the input of the neuron if the disturbance is small enough. Then, assuming linear neurons and using the chain rule, one can find the derivative of error with respect to the weights. However, if the neurons are not perfectly linear, this method will not produce the real derivatives. Another method could be adding disturbance to the weights and then finding the derivative with respect to the weights directly. However, this will increase the simulation time considerably. Thus, another method was devised: assuming that the synapses are linear

again, we applied disturbances to all the weights, which indeed produces a disturbance at the input of the neuron. The simulations show that this will improve the convergence properties of Madaline Rule III. This method is Madaline Rule III with a minor modification. We do call this method Madaline Rule III because it is in general the same algorithm. It should also be noted that the simulator normally spends most of its processing time trying to reach a DC convergence point. However, when implementing Madaline Rule III, our circuit simulator keeps the previous DC operating points in memory so that DC convergence is reached in one iteration as the disturbances in Madaline Rule III are very small.

Although we developed a method for the simulation of analog neural networks, it is not cost effective to fully simulate them using SPICE models. Therefore, before starting Madaline Rule III, an approximate starting point for the weights should be found. Approximate determination of weights is done using the modified backpropagation algorithm, which incorporates the approximate models of the synapses and neurons. For modeling the synapses and neurons three different approaches are used, namely regression by analytical functions, approximation by neural networks and approximation by look-up tables, which will be described below.

4.1 Modeling by Regression

The sigmoid, $\phi(\cdot)$, and multiplier, $\mu(\cdot,\cdot)$ are implemented using analog VLSI. Because their behavior is not known analytically, data are provided through HSpice simulations and approximated for the two functions, sigmoid and multiplier. We modeled the synapses as a polynomial function of the two variables w and x with errors of less than 8%. The polynomial used to approximate the synapse function is given as,

$$\mu(x,w) = A + Bw + Cx + Dxw + Exw^2 + Fx^2w + Gx^2w^2 + Hx^2w^3 + Kx^3w2 + Lx^3w^3 \quad (9)$$

where $\mu(x,w)$ is the output current for the input pair x,w and $A,B,...L$ are the coefficients to be determined. Data for the multiplier are taken in a small neuron structure with two synapses connected to a neuron in order to take into account the effect of the current summing circuitry.

The sigmoids are modeled in a more general form of sigmoid function, given as $f(x) = A + \frac{B}{1+e^{Cx+D}}$. The model we used has a mismatch below 2%.

4.2 Modeling by Neural Networks

Approximators used are multilayer perceptrons. We use finite differences to approximate the derivative and partial derivatives of a function.The multiplier characteristics is given in Figure 6. The approximator is a multilayer perceptron with four hidden units. This neural network is trained according to the rules given by Eqs. (5) and (6). Data obtained from the HSpice chracaterizations are used as training samples. Compared with the ideal multiplier, it has nonlinear behaviour for large input x, with the nonlinearity getting more disturbing for large weight value, w. So for good performance, first x should be preprocessed and normalized to have small magnitudes around zero.

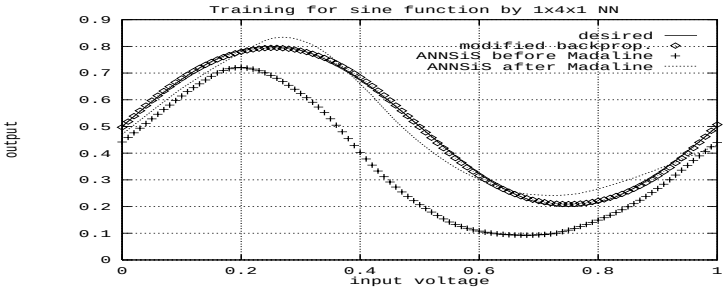

Figure 9: Sine function approximation results.

Next, as implemented using analog hardware, the nonlinearity $\phi(\cdot)$ is given in Figure 8. Note that it is similar to negative sigmoid and this is because the summation operation at the opamp introduces a negative sign which has to be cancelled. The neural network approximator to the sigmoid circuit is a multilayer perceptron with one hidden unit.

4.3 Modeling by Table Look-Up

The synapses are modeled by a look-up table of size 81×81 where the inputs and weights varying between $-2V$ and $2V$ are quantized to 81 different values each. The closest value is chosen when the inputs or the weights do not match the values in the table. Derivatives are calculated from the finite difference equation. It is obvious that the training time required is inversely proportional and the performance is directly proportional to the size of the table. The 81×81 size was chosen as it gave superior performance to the previous two methods.

When all these models were compared, it was observed that regression gave the best results for the neuron modeling. Regression also gave good results for synapse modeling, especially for the two small examples given below. However, it appears that modeling by table look-up will yield better results for larger networks.

5 Conclusion

The training method is tested on many examples, two of which are the classical XOR problem and a sine function generator. For the XOR problem, a 2x3x1 structure was used, while a 1x4x1 structure was employed for the sine generator. The weights are calculated via modified backpropagation using the models for the synapse and neuron circuits and the error decreased below 1% for both cases. At this time, the network was simulated by ANNSiS and the error was found to be 13% for the XOR example and 44% for the sine generator example. These errors were larger than the errors estimated by the backpropagation algorithm. This shows that, even in simple examples, small model mismatches (less than 10%) can create big problems. After that we applied Madaline Rule III for several epochs and the error obtained using the simulator decreased below 1% for the XOR case and below 9% for the sine generator case. Figure 9 shows the sine function approximation results obtained from training on ANNSyS.

In this study, An Analog Neural Network Synthesis System (ANNSyS) was developed. This package consists of an Analog Neural Network Simulation System (ANNSiS), a function approximator for synapses and neurons, a modified backpropagation algorithm, and a circuit level neural network trainer using modified Madaline Rule III as well as a control shell. ANNSiS is based on circuit partitioning techniques and has superior performance compared to other circuit simulators for simulating analog neural networks. ANNSyS has been applied to a number of neural network problems several of which have been illustrated in this paper and excellent results have been obtained.

References

[1] A-J. Annema, *Feed-forward Neural Networks, Vector Decomposition Analysis, Modelling and Analog Implementation,* Kluwer Academic Publishers, Boston 1995.

[2] H. Binici, G. Dündar, and S. Balkır, "A new multiplier architecture based on radix-2 conversion scheme," *Proceedings of ECCTD'95*, pp 439-442, Istanbul, 1995.

[3] P. Treleaven, M. Pacheco, and M. Vellasco,"VLSI architectures for neural networks," *IEEE Micro. Mag.,* pp 8-27, Dec 1989.

[4] Intel 80170NX ETANN Data Sheets, Feb. 1991.

[5] O. Rossetto et al., "Analog VLSI synaptic matrices as building blocks for neural networks," *IEEE Micro Mag.,* pp 56-63, Dec 1989.

[6] G. Dündar and K. Rose, "Analog neural network circuits for ASIC fabrication," *Proc. of the 5th. IEEE ASIC Conference,* Rochester, 1992, pp 419-422.

[7] G. Dündar, F-C. Hsu, and K. Rose, "Effects of nonlinear synapses on the performance of multilayer neural networks," *Neural Computation,* Vol.8, No:5, pp. 939-949, July 1996.

[8] A. Şimşek, M. Civelek, and G. Dündar, "Study of the effects of nonidealities in multilayer neural networks with circuit level simulation," *Proc. of Melecon'1996,* Bari, Vol.1, pp. 613-616, May 1996.

[9] G. Dündar and K. Rose, "The effects of quantization on multilayer neural networks," *IEEE Transactions on Neural Networks,* Vol.6, No. 6, pp. 1446-1451, Nov. 1995.

[10] B. Widrow and M. Lehr, "30 years of adaptive neural networks: Perceptron, Madaline, and Backpropagation," *Proc. of IEEE,* Vol.78, No:9, pp. 1415-1442, Sept. 1990.

[11] B. Gilbert, "A precise four-quadrant multiplier with subnanosecond responce," *IEEE Journal of Solid State Circuits,* Vol. 3, pp. 365-373, 1968.

[12] F.J. Kub et al., "Programmable analog matrix multipliers," *IEEE Journal of Solid State Circuits,* Vol. 25, pp. 207-214, 1990.

[13] T. Shima et al., "Neurochips with on-chip backpropagation and/or Hebbian learning," *IEEE Journal of Solid State Circuits,* Vol. 27, pp. 1868-1876, Dec. 1992.