PERGAMON

Neural Networks 12 (1999) 325–338

Neural Networks

Contributed article

ANNSyS: an Analog Neural Network Synthesis System

İsmet Bayraktaroğlu[a], Arif Selçuk Öğrenci[b], Günhan Dündar[b,*], Sina Balkır[b], Ethem Alpaydın[c]

[a]*Computer Science and Engineering Department, UC San Diego, San Diego, CA 92093, USA*
[b]*Department of Electrical and Electronic Engineering, Boğaziçi University, Bebek 80815, Istanbul, Turkey*
[c]*Department of Computer Engineering, Boğaziçi University, Bebek 80815, Istanbul, Turkey*

Received 19 January 1998; accepted 3 August 1998

Abstract

A synthesis system based on a circuit simulator and a silicon assembler for analog neural networks to be implemented in MOS technology is presented. The system approximates on-chip training of the neural network under consideration and provides the best starting point for 'chip-in-the-loop training'. Behaviour of the analog neural network 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. In this stage, the simulator has been combined with Madaline Rule III for approximating on chip training by software, thus minimizing the effects of circuit nonidealities on neural networks. The circuit simulator partitions the circuit into decoupled blocks which can be simulated separately, with the output of one block being the input for the next one. Finally, the silicon assembler generates the layout for the neural network by reading analog standard cells from a library. The system's performance has been demonstrated by several examples. © 1999 Elsevier Science Ltd. All rights reserved.

Keywords: Neural network implementations; Multilayer perceptrons; Circuit simulation; Madaline rule; CMOS neural networks; Neural network training; Macromodeling

Nomenclature

μ	Synapse function
x	Input
w	Weight
ϕ	Sigmoid function
y	Output of a multilayer perceptron
H	Output of a hidden unit
T	Weights in the second layer
W	Weights in the first layer
χ	Training set
r	Desired output
p	Pattern number
E	Squared error
η	Learning rate
λ	Weight decay coefficient

1. Introduction

Neural networks have gained popularity in the last few years due to their success in diverse applications. 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, they are suitable for VLSI implementation. These implementations may be digital or analog. Many digital implementations have been reported in the literature owing to the fact that they offer several advantages such as predictable accuracy, high noise immunity, ease of multiplexing communication and computation, availability of well-established tools for digital design, and ease of interfacing with other digital systems (Beiu, 1997). Analog implementations, on the other hand, have many advantages such as small size, high speed, and straightforward interfacing with the outside world which is analog by nature (Annema, 1995).

Synapses, which are the most common elements in a neural network, can be represented at the circuit level by multipliers. Parallel digital multipliers require a very large area compared to analog multipliers of comparable precision which use less than 20 transistors. For instance, parallel digital multipliers of 8×8 input word lengths have transistor counts of the order of at least several thousand (Binici et al., 1995). Serial digital multipliers are smaller than their parallel counterparts; however, they are much slower. On the other hand, the speed of an analog multiplier is limited mostly by its settling time. When one looks at neurons, a similar picture can be seen. Again, an adder and nonlinearity can be realized by less than 20 transistors in the analog

* Corresponding author. Tel.: +90-212-263-1540 Ext. 2205; Fax: +90-212-287-2465; E-mail: dundar@boun.edu.tr

PII: S0893-6080(98)00122-1

domain, whereas the same operations require transistor counts which are at least an order of magnitude larger in the digital domain assuming that multiple input parallel adders and efficient look-up tables for the nonlinearity are utilized. (Treleaven et al., 1989; Intel, 1991; Rossetto et al., 1989; Dündar and Rose, 1992).

However, analog neural network implementations have been rather ad hoc in that very few, if any, have explored the constraints that circuit non-idealities bring about. Examples are nonlinear synapses (Dündar et al., 1996; Şimşek et al., 1996), neurons that deviate from ideal functions, or errors and limitations in storing weights (Dündar and Rose, 1995; Dolenko and Card, 1995; Piché, 1995). It has previously been shown (Hollis and Paulos, 1990; Lont and Guggenbühl, 1992; Edwards and Murray, 1996; Dündar et al., 1996; Şimşek et al., 1996) that for many applications, multiplier nonlinearity can be a very severe problem even for nonlinearity factors of less than 10%. Limited precision in storing weights has also proven to be a crucial problem in analog neural network design. The work in this area has been mostly limited to predicting these effects either through simulation or through theoretical analysis, and developing some methods to partially overcome these problems. Şimşek et al. (1996) studied the effects of some non-idealities 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 will be increased 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. These approaches can be summarized as follows (Annema, 1995):

- Non-learning network. In this method, the weights are hardwired through the implementation of the fixed gain multiplier. In this case, the weights to be hardwired must be calculated before the operation of the neural network. The calculations are done on a computer which uses the model of the analog neural network. The performance of this method depends heavily on the matching between the model and the real circuit, which is a task that is very difficult to achieve.
- Neural networks in analog hardware implementation with externally adjustable weight construction. For this realization, the weights are again computed on a host computer and downloaded to the chip. Then the weights are fine tuned. The chip is used for forward pass, host computer is used for feedback (weight adaptation). In this way, the matching of the model and analog hardware is considerably increased.
- Neural network with on-chip learning. In this scheme, both the feedforward structure and all circuitry required to adapt the weights are realized on the chip. A major disadvantage of this approach and the previous one is that they both require additional hardware which is used only at the training stage.

An implementation of a general purpose analog VLSI neural network with on-chip learning has been presented in Montalvo et al. (1997a, 1997b). One commercial implementation of analog neural networks is the ETANN 80170NX chip (Intel, 1991). 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 (Widrow and Lehr, 1990) has been suggested for the ETANN chip; however, this requires a host computer and external hardware besides there being many timing problems which limit the performance of training. Problems like these have prevented the success of this chip on the market so that commercial applications using this chip and similar ones have been limited. Several applications reported in the literature have demonstrated successful operation, whereas other reported applications have suffered from the aforementioned problems. Another major deficiency of this chip is the issue of cyclability in the weight storing EAROMs. Therefore the number of iterations required for chip-in-the-loop training has to be minimized or eliminated if possible. This can be achieved by having a suitable initial weight set.

The best approximation to a circuit is its SPICE model. The difference between the actual chip and its SPICE model is mainly due to variations in process parameters over which the user has no control and most often no a priori information. For this purpose, the approximation of on-chip training by software using circuit models of the actual nonideal synapse and neuron circuitry is proposed and a special circuit simulator has been developed. The proposed system approximates on-chip training of the neural network under consideration and provides the best starting point for 'chip-in-the-loop training'. Moreover, starting chip-in-the-loop or on-chip training from scratch will introduce convergence problems, since the error surface will have many spurious local minima and plateaus. In the approach presented in this paper, the nonidealities are incorporated using soft constraints; that is, we start gradient descent assuming ideal components and thereby converge rapidly to the relevant part of the weight space. Afterwards, we switch to the analytical models of the nonideal components and also introduce weight decay to keep the weight values small by adding a soft penalty term to the error function. The final part of the training is performed on the circuit simulator using Madaline Rule III and it follows that the best starting point is achieved with minimal convergence problems.

The outline of this paper is as follows. Section 2 introduces the Analog Neural Network Simulation System (ANNSiS) which is based on circuit partitioning techniques. This system is used to simulate neural networks composed of the building blocks discussed in Section 3 at circuit level.

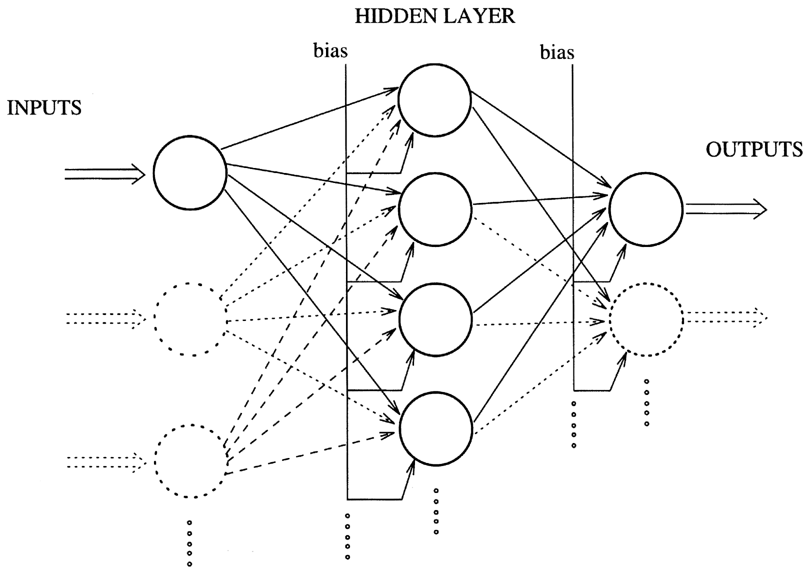

Fig. 1. The general structure of the multilayer perceptron.

Modeling those nonideal components for best performance is discussed in Section 4. Training on ANNSyS is proposed in Section 5. SAFANN: Silicon Assembler For Analog Neural Networks, which generates the layout of the analog neural network for VLSI realization, is introduced in Section 6, with sample layouts generated by SAFANN. The design examples are discussed in Section 7. Finally, Section 8 concludes the paper.

2. ANNSiS: Analog Neural Network Simulation System

The general structure of a multilayer perceptron with a single hidden layer is given in Fig. 1. Each neuron can be represented as a collection of three main blocks, namely synapses, adders and a nonlinearity. The synapse function can be denoted as $\mu(w,x)$ where w and x represent the weight and the input connected to the synapse, respectively.

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 by 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, i.e. the outputs of the neurons are not loaded by the inputs

Table 1
Simulation results of various neural network architectures with SPICE2G6 and ANNSiS

				SPICE2G6		ANNSiS			
						Without partitioning		With partitioning	
Structure	Synapses + opamps	FETs	Nodes	Memory (k)	Time (s)	Memory (k)	Time (s)	Memory (k)	Time (s)
2×1	3	49	148	672	3.6	448	0.8	560	0.8
$2 \times 2 \times 1$	9	147	426	908	13.3	596	2.7	812	1.7
$2 \times 3 \times 1$	13	213	611	980	19.6	672	4.3	956	2.4
$2 \times 4 \times 1$	17	279	796	1040	26.3	748	6.6	1084	3.0
$2 \times 5 \times 1$	21	345	981	1100	31.9	820	8.8	1208	3.7
$2 \times 5 \times 2$	27	445	1258	1212	49.3	928	14.5	1400	5.1
$2 \times 5 \times 3$	33	545	1535	*	*	1028	20.6	1564	6.3
$2 \times 5 \times 4$	39	645	1812	*	*	1128	26.6	1696	7.0
$2 \times 5 \times 5$	45	745	2089	*	*	1264	32.3	1912	8.8
$4 \times 8 \times 7$	103	1721	4766	*	*	2136	128.0	3112	19.4

* SP1CE2G6 did not converge up to the predefined number of iterations of SPICE which is 10000.

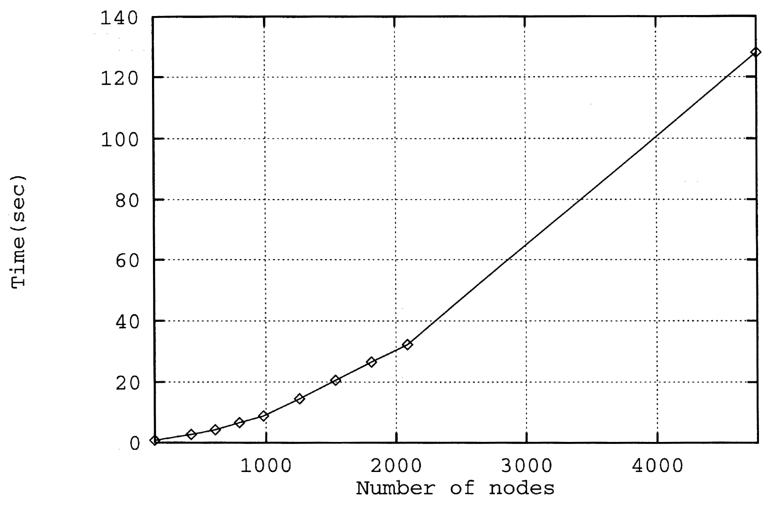

Fig. 2. Simulation time for different MLP structures (without partitioning).

of the synapses of the next layer, the circuit can be partitioned into blocks which can be simulated separately starting from the input layer. Most of the neural network implementations in the literature use CMOS technology. Considering this technology, the assumption of being completely decoupled holds and allows partitioning of 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. In this way, the input is propagated to the output.

Different sized analog neural network structures were first created with the building blocks described in Section 3, namely, Synapses (current output Gilbert multiplier), OPAMPs (used as an I–V converter), and Sigmoid generators (nonlinearity for the neurons). These structures were first simulated without partitioning by SPICE2G6 and ANNSiS. Next, they were simulated using ANNSiS by partitioning into blocks which consisted of a neuron and all synapses connected to that neuron. Simulation results of SPICE2G6 and ANNSiS were compared for accuracy and found to be exactly matching. Table 1 shows the sizes, CPU times, and the memory requirements for different structures for simulations performed on SunSPARC2 workstations.

Various MLP structures were simulated both with and without partitioning using ANNSiS. As seen in Figs. 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. Implementation of analog neural network circuitry

Several representative circuits are described in this

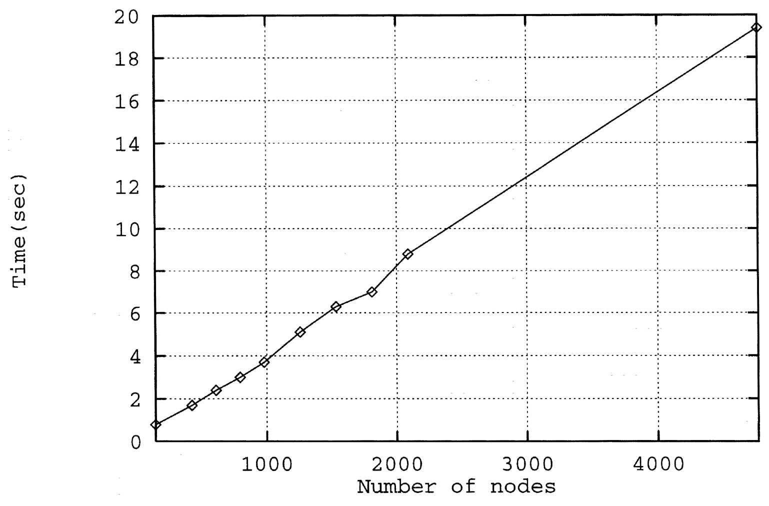

Fig. 3. Simulation time for different MLP structures (with partitioning).

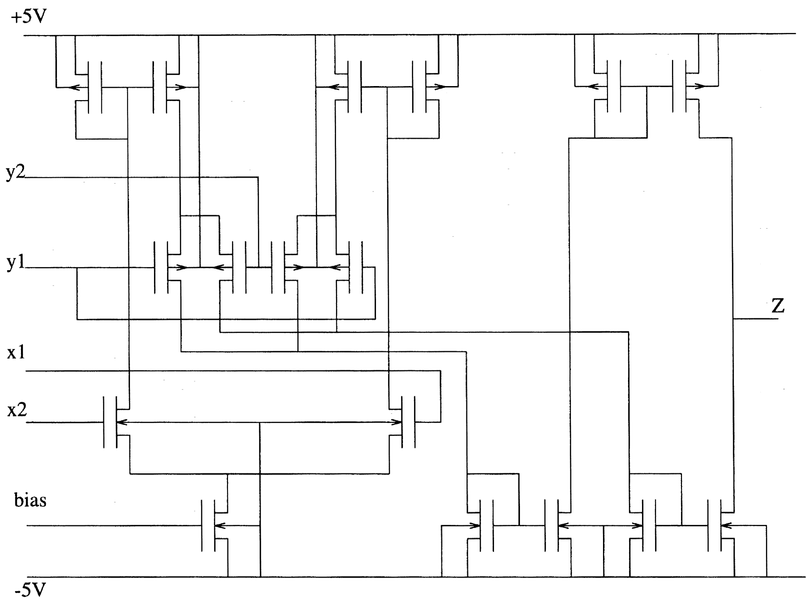

Fig. 4. Four-quadrant Gilbert multiplier.

section for demonstrating the proposed approach. The approach is not limited to the circuits described here, but can be used with other synapse and neuron implementations.

The synapses in a neural network can be realized by analog multipliers if the inputs and the weights can be represented by voltages. An illustrative synapse circuit which is a modified version of the well known Gilbert multiplier (Gilbert, 1968; Kub et al., 1990) is shown in Fig. 4. The inputs are in the form of voltage differences and are denoted by $x_1 - x_2$ 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 (for different weight values, $w = \{-2,-1,0,1,2\}$ volts) is shown in Fig. 5(b).

The use of current-output synapses enables the summation of those currents by simply connecting them together in a neuron. This current sum can be converted to a voltage by using an OPAMP as a current-to-voltage converter. The general structure of the analog neural network cell can be seen in Fig. 6. Fig. 7 shows an OPAMP that can be used for the above mentioned purpose. 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 similar to the synapse circuit above. A sigmoid generator introduced in (Shima et al., 1992) is used after the OPAMP to generate the activation function for the neuron. This generator is depicted in Fig. 8, and the characteristics are plotted in Fig. 9. Layouts of the analog neural network circuitry described above are designed in a full-custom manner based on 2.4 μm CMOS technology. The netlists are extracted from the layouts and simulated for characterization purposes using SPICE. It is evident from those characteristics that the actual circuits exhibit nonideal behaviour so that an adaptation in the training algorithm is required. As seen in Fig. 5(b), the nonideality is best visible in the synapse characteristics where the output current curves deviate highly from linearity for large values of inputs and/or weights.

4. Modeling nonideal components

Since a tool optimized for simulating analog neural networks at the circuit level has been developed, it can be used in conjunction with Madaline Rule III (Widrow and Lehr, 1990) for training purposes. However, the computational complexity of such a training technique renders it unuseable for most practical problems. In order to reduce the computational complexity, the number of Madaline Rule III iterations must be kept as low as possible. Hence, an appropriate weight set must be obtained prior to circuit level training. This can be achieved by backpropagation using relatively simple macromodels for synapses and neurons.

For macromodeling the synapses and neurons, three different approaches were used, namely, regression by analytical functions, approximation by neural networks, and representation by look-up tables which will be described below.

4.1. Modeling by regression

Analytical expressions for the functional behavior of the synapse and neuron circuits are not available. Hence, the

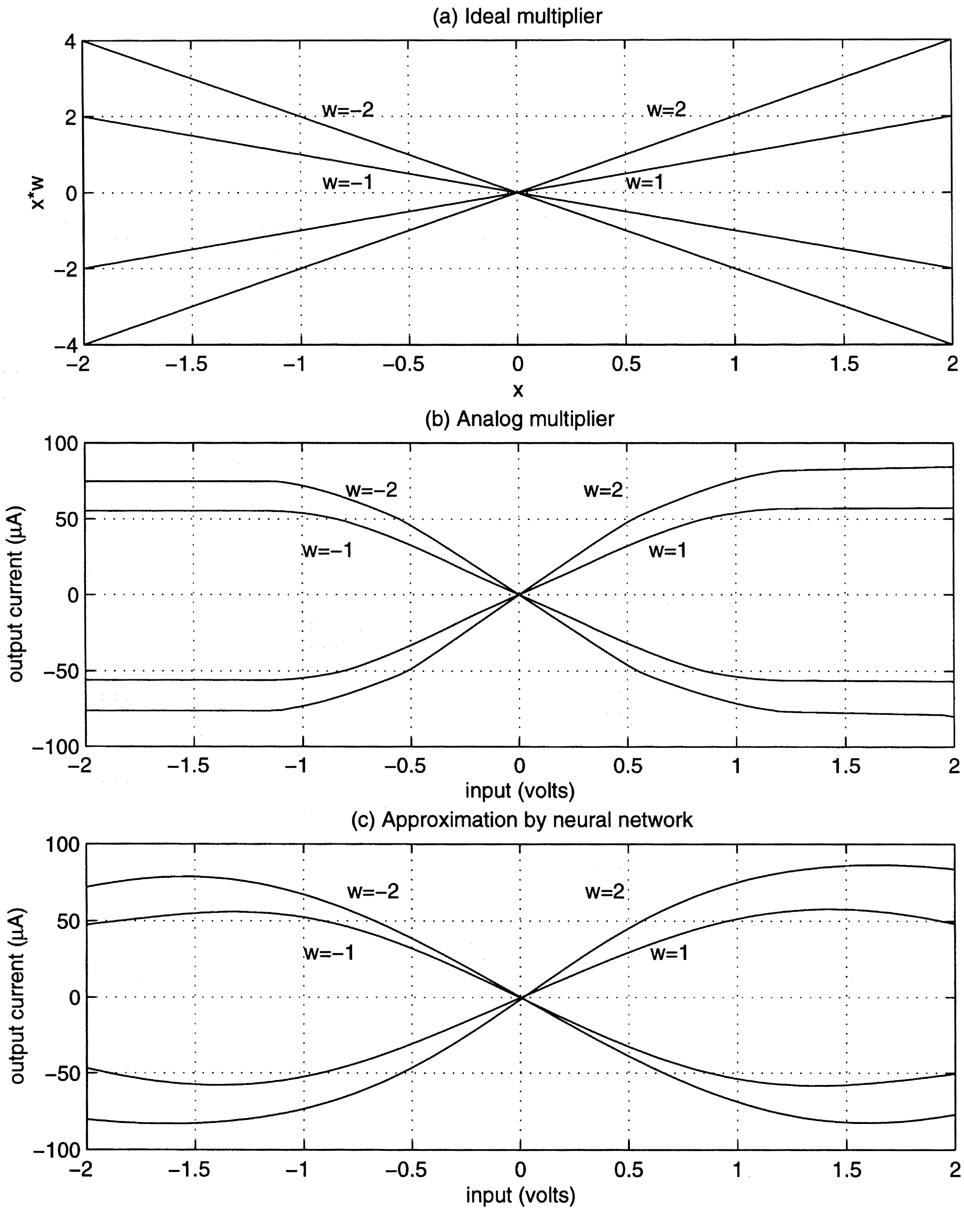

Fig. 5. (a) Ideal multiplier; (b) Multiplier data provided by SPICE simulation; (c) Neural network approximation to (b).

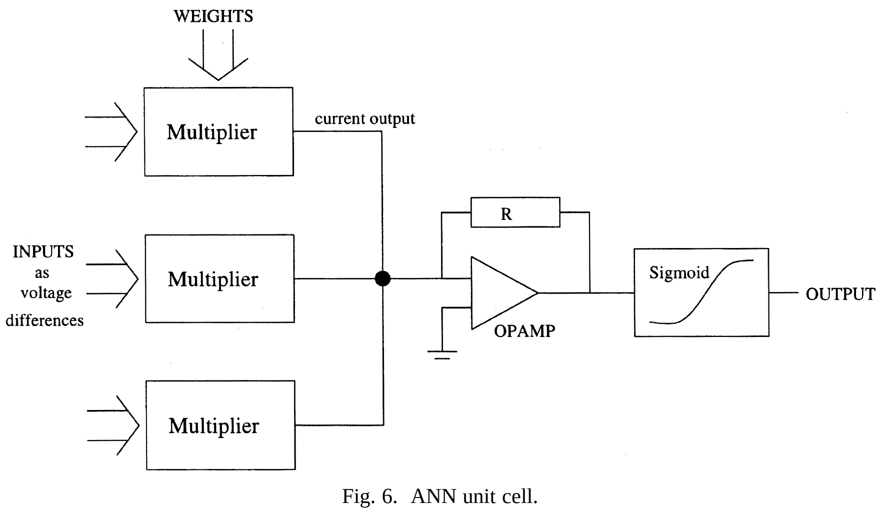

Fig. 6. ANN unit cell.

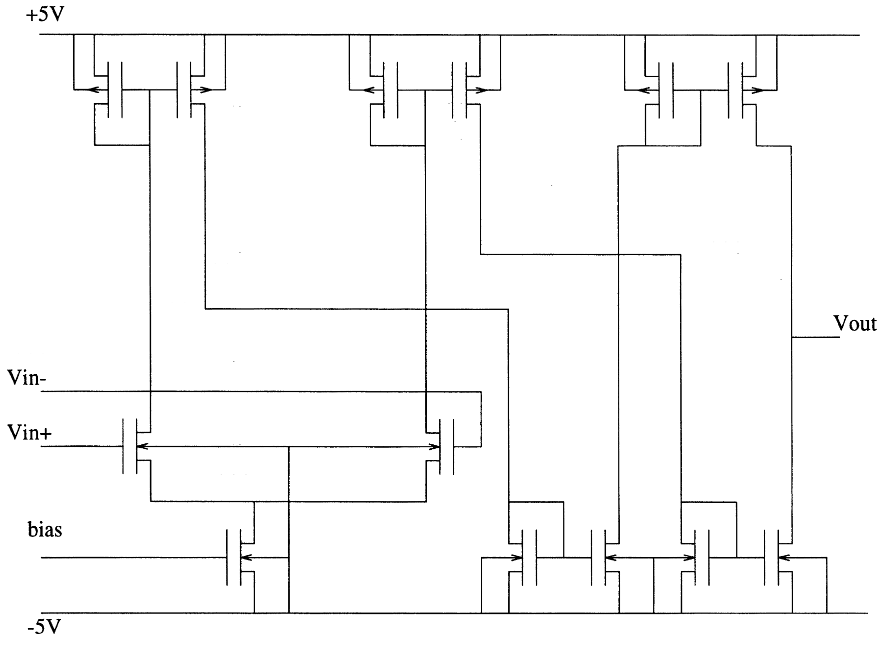

Fig. 7. General purpose OPAMP.

synapses were modeled as a polynomial function of the two variables w and x with errors of less than 8% based on data obtained from SPICE simulations. The polynomial used to approximate the synapse function is given as,

$$\mu(x,w) = A + Bw + Cx + Dxw + Exw^2 + Fx^2w + Gx^2w^2 + Hx^2w^3 + Kx^3w^2 + Lx^3w^3 \quad (1)$$

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 the effect of the current summing circuitry into account. The values for the coefficients A,B,…,L found by regression using the method of *least-squares fit* are given in Table 2. Their normalized values with respect to D are also given. It is evident that the coefficient D for the linear term dominates for small x,w values whereas the coefficient L for the cubic term comes next in effect. Hence, it can be concluded that the nonlinearity of the multiplier is mainly in cubic terms.

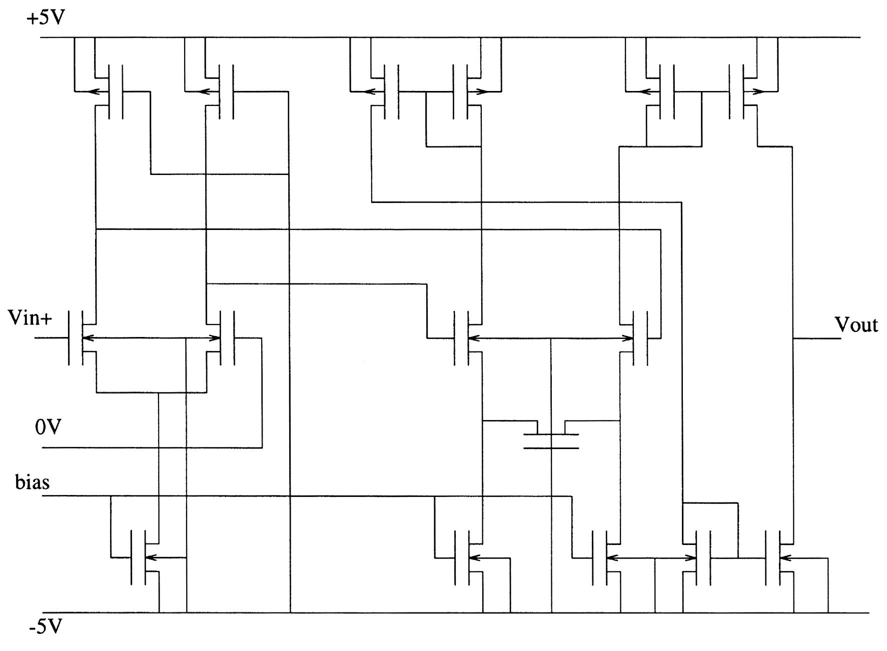

Fig. 8. Sigmoid generator.

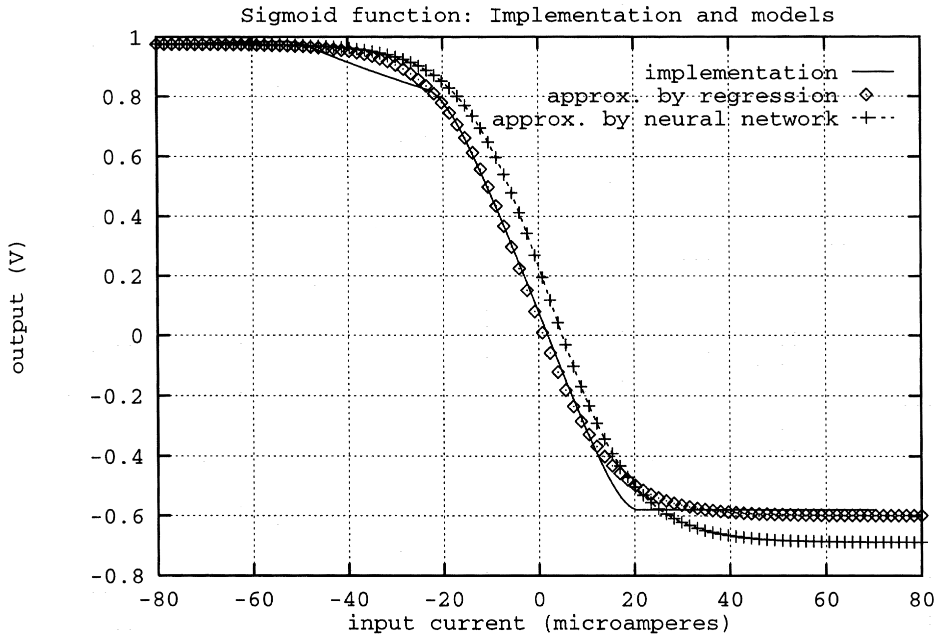

Fig. 9. Nonideal sigmoid implemented and its approximation by regression and 1 × 1 × 1 neural network.

The sigmoid and the current-to-voltage conversion circuitry are modeled together as a single unit, in a more general form of sigmoid function, given as

$$\phi(x) = A + \frac{B}{1 + e^{Cx+D}} \tag{2}$$

where x denotes the sum of the currents from the synapses connected to the neuron and $\phi(x)$ is the output of the neuron as a voltage. The model used has an error below 5%. The values for the coefficients are as follows: $A = -0.5986$; $B = 1.5745$; $C = 1.1462 \times 10^5$; $D = 0.3711$. Partial derivatives of current ($\partial\mu/\partial x$, $\partial\mu/\partial w$) and ϕ' can be calculated directly using Eqs. (1) and (2). The nonlinearity $\phi(x)$ as implemented and modeled can be seen in Fig. 9. Note that it is a negative sigmoid because the summation operation at the OPAMP introduces a negative sign which has to be cancelled.

Table 2
Regression coefficients for the multiplier

Coefficient	Value ($\times 10^{-6}$)	Value normalized with respect to *D*
A	0.00029	0.0007
B	−0.00182	−0.0042
C	0.00095	0.0022
D	0.43850	1.0000
E	−0.00443	−0.0100
F	0.00132	0.0030
G	−0.00099	−0.0023
H	0.00092	0.0021
K	0.00056	0.0013
L	−0.01864	−0.0425

4.2. Modeling by neural networks

Approximators used are multilayer perceptrons. Finite difference was used to calculate the derivatives of the approximators. The same technique was also used to approximate a partial derivative.

The multiplier characteristics are given in Fig. 5(b). The approximator is a multi-layer perceptron with four hidden units. This neural network is trained using standard back-propagation. Data obtained from the SPICE characterizations of the netlists extracted from the layout are used as training samples. Compared with the ideal multiplier, it has nonlinear behaviour for large input x, with the nonlinearity becoming pronounced for large weight values (Fig. 5(c)). The neural network approximator to the sigmoid circuit is a multilayer perceptron with one hidden unit, the model for the nonlinearity $\phi(x)$ is given in Fig. 9.

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 −2 V and 2 V are quantized to 81 different values, each centered around 0 V with 40 values residing on both positive and negative voltage axes. The closest value is chosen when the inputs or the weights do not match the values in the table. Derivatives are computed from finite differences. It is obvious that the training time required is inversely proportional and the approximation 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. It was observed that larger look-up tables yielded long simulation times with marginal improvements in approximation performance, whereas the weak approximation performance of smaller tables made this method inapplicable.

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 small examples. However, it appears that modeling by table look-up will yield better results for larger networks employing many synapses.

5. Training using Madaline Rule III

For simplicity of notation, a scalar input-output case will be considered. (Solid drawings in Fig. 1.) Extension to multidimensional inputs and outputs is straightforward. The output of a multilayer perceptron (y) is a weighted sum of the outputs of a number (h) of hidden units, H_i, filtered through a nonlinear function $\phi(x)$:

$$y(x) = \phi\left(\sum_{i=1}^{h} \mu(T_i, H_i(x)) + \mu(T_0, 1)\right), \tag{3}$$

where x is the input, and T_i are the weights associated with the outputs of hidden units. T_0 is the bias weight. Output of each hidden unit is another similar sum:

$$H_i(x) = \phi(\mu(W_i, x) + \mu(W_{i0}, 1)), \tag{4}$$

where W_i are the weights associated with the hidden units. W_{i0} are the bias weights for the units in the hidden layer.

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

$$E(W, T) = \sum_p E^p = \sum_p [r^p - y(x^p)]^2. \tag{5}$$

In the backpropagation algorithm, parameters W,T are updated to minimize E using gradient-descent, where in the online version, for each pattern pair, the following updates are done:

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

where η is the learning rate. So the learning method can be implemented using any synapse and nonlinearity if $\mu(w,x)$, $\partial\mu/\partial x$, $\partial\mu/\partial w$, ϕ and ϕ' are given. However, describing the behaviour of the synapses and neurons analytically is not a simple task, as shown in the modeling section. In the ideal case, the synapse performs the multiplication of the input and the weight, i.e., $\mu(w,x) = w \cdot x$, $\partial\mu/\partial x = w$ and $\partial\mu/\partial w = x$. Most frequently, the nonlinearity is the sigmoid function:

$$\phi(a) = \frac{1}{1 + \exp(-a)}. \tag{7}$$

Then, the output becomes:

$$y(x) = \phi\left(\sum_{i=1}^{h} T_i H_i(x) + T_0\right). \tag{8}$$

Consequently, we get the classical update equations for the backpropagation algorithm by rewriting Eq. (6):

$$\begin{aligned}
\Delta T_i &= \eta\frac{\partial E^p}{\partial T_i} = \frac{dE^p}{dy^p}\frac{\partial y^p}{\partial T_i} = \eta(r^p - y^p)H_i(x^p) \\
\Delta T_0 &= \eta(r^p - y^p) \\
\Delta W_i &= \eta(r^p - y^p)T_i H_i(x^p)(1 - H_i(x^p))x^p \\
\Delta W_{i0} &= \eta(r^p - y^p)T_i H_i(x^p)(1 - H_i(x^p)),
\end{aligned} \tag{9}$$

where $\phi'(x^p) = \phi(x^p)(1 - \phi(x^p))$ if $\phi(x)$ is the sigmoid function, as given in Eq. (7).

As explained in Section 4, the approximate weights are calculated with the backpropagation algorithm as modified according to Eqs. (3) and (6) using the table look-up model for the synapses and the regression model for the sigmoidal block. Given a training set $\chi = \{x^p, r^p\}_p$ and a certain network topology, the input–output values need to be scaled such that they fall within the operational range of the analog circuitry. That is, input values have to be within the linear region (-2 V to $+2$ V) of the multipliers, and the output values have to be within the output range (-0.6 V to $+1$ V) of the sigmoid function generator. At this stage, a further modification in the update rules is applied to favor small weight values so that the synapses operate in their linear region. This is done by the *weight decay* technique. In weight decay, the error function Eq. (5) is modified to be:

$$E(W, T) = \sum_p [rp - y(x^p)]^2 + \lambda\sum_i w_i^2 \tag{10}$$

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:

$$\Delta w_i = -\eta\frac{\partial E}{\partial w} - \lambda w_i. \tag{11}$$

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. (10) is the maximum a posteriori solution (Bishop,

Table 3
Madaline Rule III training algorithm

```
while (($error > tolerance$) and
        ($no\_of\_iteration < iteration\_limit$))
{
for (index1 = 1, $no\_of\_samples$)
        {
        simulate the network using ANNSiS,
        compute $error$;
        for (index2 = 1, $no\_of\_neurons$)
        {
        add disturbance at the input of $neuron$[index2];
        simulate the network using ANNSiS,
        compute $error\_disturbed$;
        compute $error\_difference$, weight updates,
        $delta\_w$[index2,index1], for $neuron$[index2];
        $w\_update$[index2] = $w\_update$[index2]
        + $delta\_w$[index2,index1];
        }
        }
        update the weights;
        $no\_of\_iteration$ ++;
}
```

1995). 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.

These weights are then downloaded to the circuit level representation of the network to start Madaline Rule III. In order to closely approximate the analog neural network hardware, the circuit level representation is based on the extracted layout of the neural chip. Madaline Rule III is an algorithm that can be used when analytical expressions for the derivatives of error with respect to weights are not available or the expressions present excessive error. 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, the derivative of the error with respect to the input of the neuron is found if the disturbance is small enough. 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 where a Madaline iteration consists of the following steps. For each training sample, a small amount of current is injected into each neuron sequentially via a current source. That is, starting with the first neuron of the hidden layer, all neurons will be disturbed one by one where the same training sample will be applied as the input. At each disturbance the circuit is simulated and the difference in the error at the output as compared to the undisturbed case will be found. The derivatives of error with respect to the weights are calculated using the models of the neurons and the synapses (as

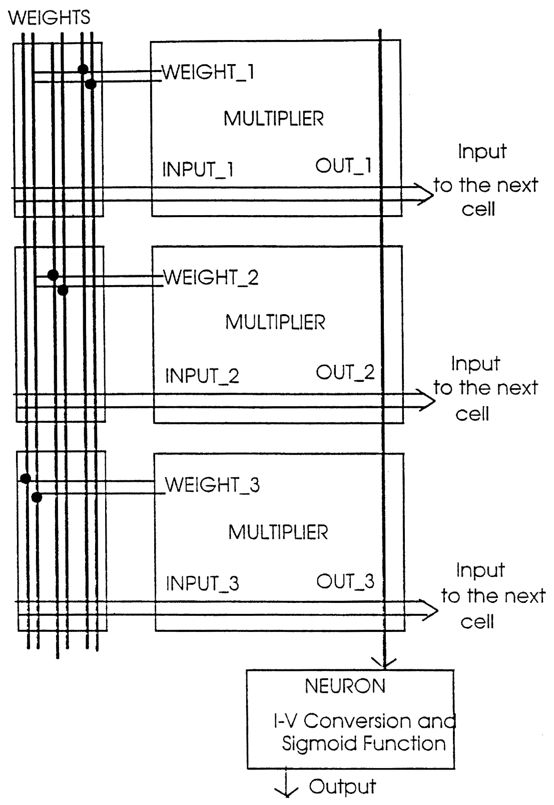

Fig. 10. Topology of a single cell with three inputs.

described in the previous section). However, weights will not be updated until all patterns are applied, that is, batch update is performed on the weights where the stored weight updates are added algebraically. This method is still referred to as Madaline Rule III because it is a reformulation of 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, the proposed 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. The computational time required for one Madaline iteration was observed to be approximately twice that of one forward simulation by ANNSiS. The Madaline iterations continue until the error decreases below a user-defined value, or until a maximum iteration count is exceeded. The flow of the above algorithm is summarized in Table 3.

6. Silicon assembler

The automatic generation of the analog neural network layout is possible if there are suitably designed synapses and neurons. These basic blocks (subcells) can be placed in arrays and the complete circuit layout will be obtained. As part of this study, a silicon assembler (SAFANN) was developed to accomplish this task.

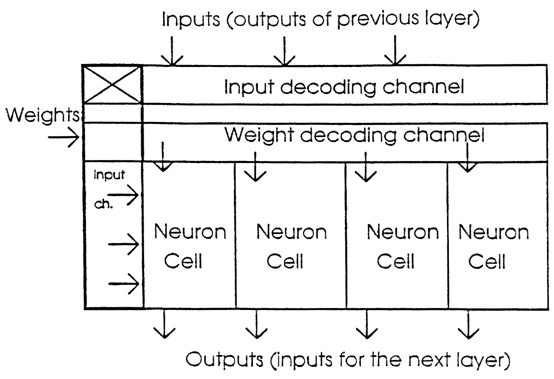

Fig. 11. Topology of a single layer with three inputs and four neurons.

The starting point is the layout for a single cell which consists of three types of subcells and some interconnections. A sample 3-input cell structure is given in Fig. 10. It is mainly made up of three multipliers, a neuron and three of the so-called channels. In fact, each channel consists of three subchannels employing two lines each. There are several reasons for selecting this topology. First, the design has to be modular; that is, it should support any number of inputs. Hence, the weights for the inputs are carried on channels whose number can easily be manipulated. It should also be noticed that only one weight, i.e., 2 weight lines, should be connected to each multiplier, so that a decoding scheme is necessary. Next, the input lines have to travel throughout the cell in the horizontal direction because that input will also be required in the next cell placed to the right of the first one. Finally, the output lines of the multiplier have to be aligned such that they are common in the vertical direction so that they will be connected together which is also part of the abutment property. The supply and bias lines will also run through the cells for alignment.

SAFANN performs the automatic placement of the layout

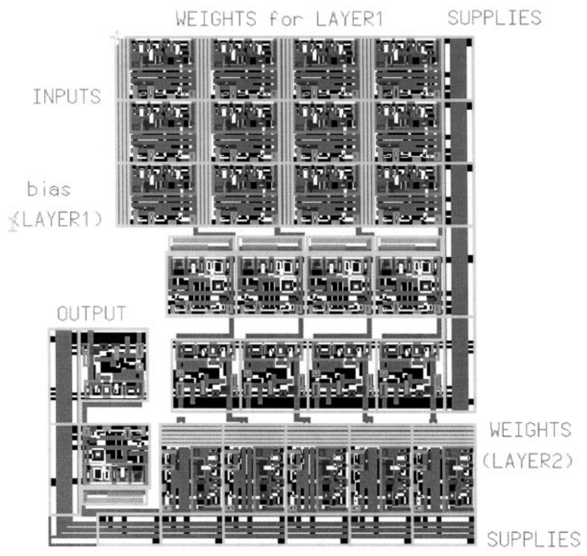

Fig. 12. Layout of XOR generated by SAFANN.

building blocks and routing between them by placing instances of symbols created for those blocks, namely the multiplier, neuron and subchannel. The structure of this methodology can be understood by considering a single layer of an analog neural network given in Fig. 11. Here, the channels will be described by collection of boxes representing metal-1, metal-2 lines and vias. Neuron cells, however will be called by the symbols. Fig. 12 presents the layout generated by SAFANN for an XOR gate consisting of two inputs, three hidden units, and an output unit. Similarly, layout generated by SAFANN for a $1 \times 10 \times 1$ structure is given in Fig. 13.

7. Numerical experiments

ANNSyS has been applied to a number of neural network problems. Two rather small examples are the classical XOR problem and a sine function generator. Recognition of spoken phonemes is also included as a large sized example.

7.1. Learning XOR problem

For the XOR problem, a $2 \times 3 \times 1$ structure was used. The weights were calculated via modified backpropagation using the models for the synapse and neuron circuits and the error decreased below 1% for the four training samples. At this time, the network was simulated by ANNSiS and the error was found to be 13% for the XOR function. Finally, Madaline Rule III was applied for 50 epochs and the error obtained using the simulator decreased below 1% again.

7.2. Learning sine function

A $1 \times 4 \times 1$ structure was employed for the sine function ($0.3 \sin(2\pi x)$) where the training set contains 20 pairs from the interval (0,1). (*desired* in Fig. 14) Again, the neural network was trained via modified backpropagation until the error decreased below 1% (*modified backprop* in Fig. 14), and then the network was simulated by ANNSiS with the weights found. The error was around 30% (*ANNSiS before Madaline* in Fig. 15). Finally, Madaline Rule III was applied for 50 epochs and the error decreased below 3% (*ANNSiS after Madaline* in Fig. 15). The result of circuit simulation with weights obtained via the standard backpropagation training assuming ideal components, i.e., linear synapses and ideal sigmoid function, is also included (*backprop. with ideal components*) in Fig. 14 to show that macromodeling is necessary for proper operation.

7.3. Speech phoneme recognition

For /b, d, g, m, n, N/ speech phoneme recognition (SR) experiments, the database contains 5240 Japanese isolated words and phrases (Alpaydin and Gürgen, 1998). Two hundred samples for each class are taken from the even-numbered and odd-numbered words. 600 samples are used for training and 600 for testing. Phonemes are extracted

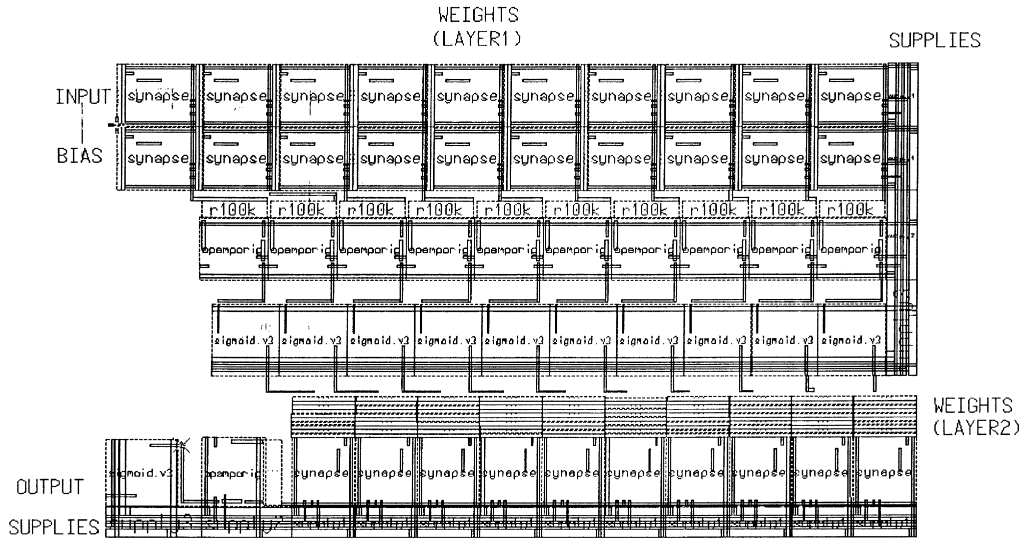

Fig. 13. Layout of 1 × 10 × 1 structure generated by SAFANN.

from hand-labeled discrete word utterances and phrase utterances which have a sampling frequency of 12 KHz. Seven speech frames (each 10 ms) are used as input. For each 10 ms frame, 16 Mel-scaled FFT coefficients are computed as feature values. This 112 dimensional input is then reduced to 21 using principal components analysis that explains 98.1% of the variance. A 21 × 8 × 6 neural network structure is employed for the training. There were 600 samples and the modified backpropagation algorithm accomplished a correct classification rate of over 90% by the software. However, when the same neural network circuitry is simulated by ANNSiS, the correct classification rate dropped to 87% and finally, the neural network is trained by the Madaline approach and the correct classification rate increased to 92%. The results from the training and test samples are displayed in Table 4.

8. Conclusion

In this study, an Analog Neural Network Synthesis System (ANNSyS) was developed. This package consists of an Analog Neural Network Simulation System (ANNSiS), a Silicon Assembler For Analog Neural Networks (SAFANN), a function approximator for synapses and neurons, a modified backpropagation algorithm, and a

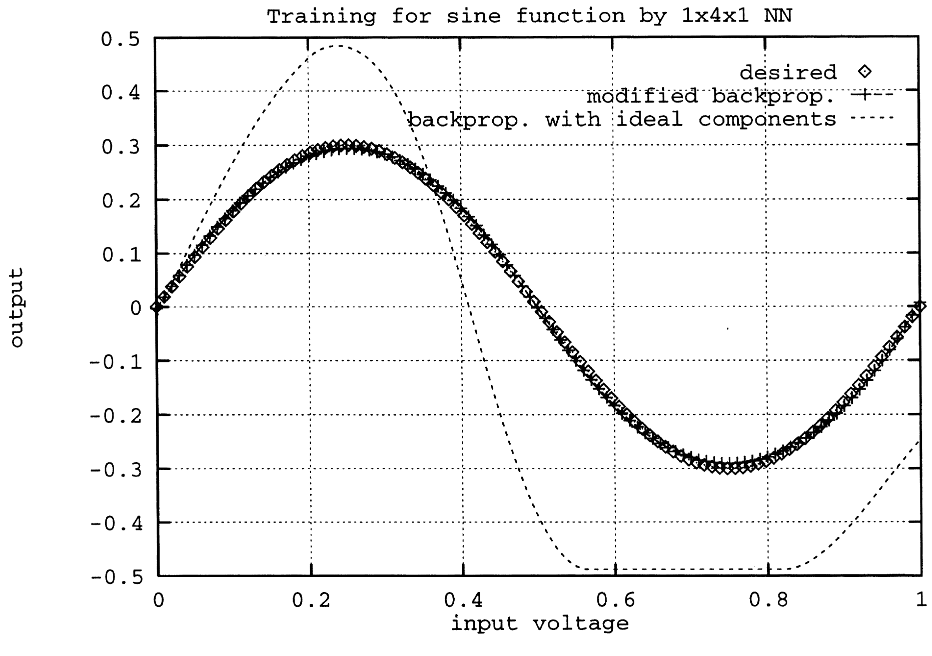

Fig. 14. Training results for sine function (Part 1).

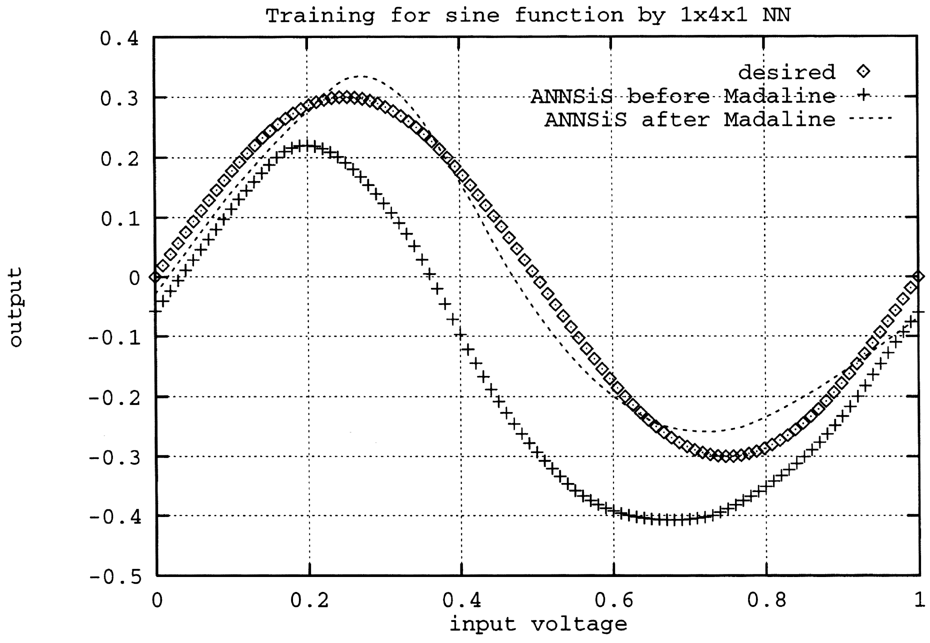

Fig. 15. Training results for sine function (Part 2).

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.

In this work, a new approach to designing and training analog neural networks was presented. Analog synapse and neuron circuitry were designed and these were modeled analytically by various methods. Although the models looked adequate by themselves, the performance of neural networks trained using these methods were well below expectations. The errors obtained by ANNSiS before Madaline Rule were larger than the errors estimated by the backpropagation algorithm. This shows that, even in simple examples, small model mismatches (less than 5%) can create considerable problems. This performance loss was especially severe in function approximation applications rather than pattern classification applications. From these results, it was clear that using models for neural network components was not enough for training the neural networks and that the network itself should be used in training. However, using the neural network chip in the training loop is something that should be avoided as much as possible. Hence, a circuit level representation of the chip was used in training instead. This had previously been prohibitive due to the huge simulation times. As an example, for the speech recognition task described above, the 21 × 8 × 6 network employs more than 4300 MOS transistors so that the computational load is very high. A software package that allows one to realize this training in a reasonable computation time with the help of ANNSiS was developed, where the neural network is partitioned for simulation. For the same example, the largest partition to be simulated at one time contains 386 transistors only. Furthermore, the efficient implementation of the Madaline Rule III allows each Madaline iteration to run in approximately just twice the time required for a single forward pass calculation. For applications where high performance is desirable, the weight set obtained via ANNSyS can be utilized as the best starting point for chip-in-the-loop training given reasonably accurate device models. Furthermore, the results of training with ANNSyS can be made more robust with respect to device mismatches in the actual circuit by perturbing device sizes and threshold voltages during training.

Table 4
Speech phoneme recognition

Correct classification rate	Training set (%)	Test set (%)
Modified backpropagation	90	72
ANNSiS before Madaline	87	65
ANNSiS after Madaline	92	78

Acknowledgements

This work is supported by TÜBİTAK under grants EEEAG-183 and EEEAG-143. Furthermore, the authors wish to express their gratitude to Fikret Gürgen for the speech phoneme data set.

References

Alpaydın, E., & Gürgen, F. (1998). Comparison of statistical and neural classifiers and their applications to optical character recognition and speech classification. In C. T. Leondes (Ed.), *Neural network systems techniques and applications* (pp. 61-88). Academic Press.

Annema, A-J. (1995). *Feed-forward neural networks, vector decomposition analysis, modelling and analog implementation.* Boston, MA: Kluwer Academic Publishers.

Beiu, V. (1997). Digital integrated circuit implementations. In E. Fiesler, & R. Beale (Eds.), *Handbook of neural computation.* Institute of Physics and Oxford University Press.

Binici, H., Dündar, G., & Balkir, S. (1995). A new multiplier architecture based on radix-2 conversion scheme. *Proceedings of European Conference on Circuit Theory and Design* (pp. 439–442).

Bishop, C. (1995). *Neural networks for pattern recognition.* Oxford University Press.

Dolenko, B. K., & Card, H. C. (1995). Tolerance to analog hardware of on-chip learning in backpropagation networks. *IEEE Transactions on Neural Networks*, *6*, 1045–1052.

Dündar, G., & Rose, K. (1992). Analog neural network circuits for ASIC fabrication. *Proceedings of the Fifth IEEE ASIC Conference* (pp. 419–422).

Dündar, G., & Rose, K. (1995). The effects of quantization on multilayer neural networks. *IEEE Transactions on Neural Networks*, *6*, 1446–1451.

Dündar, G., Hsu, F-C., & Rose, K. (1996). Effects of nonlinear synapses on the performance of multilayer neural networks. *Neural Computation*, *8*, 939–949.

Edwards, P. J., & Murray, A. F. (1996). *Analogue imprecision in MLP training.* World Scientific Publishing.

Gilbert, B. (1968). A precise four-quadrant multiplier with subnanosecond response. *IEEE Journal of Solid State Circuits*, *3*, 365–373.

Hollis, P. W., & Paulos, J. J. (1990). Artificial neural networks using MOS analog multipliers. *IEEE Journal of Solid State Circuits*, *25*, 849–855.

Intel, (1991). 80170NX ETANN Data Sheets.

Kub, F. J., Moon, K. K., Mack, I. A., & Long, F. M. (1990). Programmable analog matrix multipliers. *IEEE Journal of Solid State Circuits*, *25*, 207–214.

Lont, J. B., & Guggenbühl, W. (1992). Analog CMOS implementation of a multilayer perceptron with nonlinear synapses. *IEEE Transactions on Neural Networks*, *3*, 457–465.

Montalvo, A. J., Gyurcsik, R. S., & Paulos, J. J. (1997a). An analog VLSI neural network with on-chip perturbation learning. *IEEE Journal of Solid State Circuits*, *32*, 535–543.

Montalvo, A. J., Gyurcsik, R. S., & Paulos, J. J. (1997b). Toward a general purpose analog VLSI neural network with on-chip learning. *IEEE Transactions on Neural Networks*, *8*, 413–423.

Piché, S. W. (1995). The selection of weight accuracies for Madalines. *IEEE Transactions on Neural Networks*, *6*, 432–445.

Rossetto, O., Jutten, J., Herault, J., & Kreuzer, I. (1989). Analog VLSI synaptic matrices as building blocks for neural networks. *IEEE Micro Magazine*, *9*, 56–63.

Shima, T., Kimura, T., Kamatani, Y., Itakura, T., Fujita, Y., & Iida, T. (1992). Neurochips with on-chip backpropagation and/or Hebbian learning. *IEEE Journal of Solid State Circuits*, *27*, 1868–1876.

Şimşek, A., Civelek, M., & Dündar, G. (1996). Study of the effects of nonidealities in multilayer neural networks with circuit level simulation. *Proceedings of Mediterranean Electronics Conference* (pp. 613–616).

Treleaven, P., Pacheco, M., & Vellasco, M. (1989). VLSI architectures for neural networks. *IEEE Micro Magazine*, *9*, 8–27.

Widrow, B., & Lehr, M. (1990). 30 years of adaptive neural networks: Perceptron, Madaline, and backpropagation. *Proceedings of IEEE*, *78*, 1415–1442.