Device and Circuit-Level Modeling Using Neural Networks with Faster Training Based on Network Sparsity

A. Hafid Zaabab, *Member, IEEE,* Qi-Jun Zhang, *Senior Member, IEEE,* and Michel S. Nakhla, *Senior Member, IEEE*

***Abstract*—Recently, circuit analysis and optimization featuring neural-network models have been proposed, reducing the computational time during optimization while keeping the accuracy of physics-based models. We present a novel approach for fast training of such neural-network models based on the sparse matrix concept. The new training technique does not require any structure change in neural networks, but makes use of the inherent nature of neural networks that for each pattern some neuron activations are close to zero, and hence, have no effect on network outputs and weights update. Much of the computation effort is saved over standard training techniques, while achieving the same accuracy. FET device and VLSI interconnect modeling examples verified the proposed technique.**

***Index Terms*—Modeling, neural network, optimization, simulation.**

I. Introduction

ACCURATE and fast circuit and device models are of important concern in circuit analysis and optimization problems [1], [2]. For device models, the accuracy of physics-based models of active, passive, and transmission-line elements is required, but is at the expense of much increased computational cost [3] due to the complexity of physics equations, which consist of field equations (e.g., Maxwell's equations). Circuit simulation involves: 1) solving the overall circuit equation by relating currents and voltages at each node of the circuit and 2) finding the output response of interest (e.g., delay in an interconnect network). Solving the circuit equation is central processing unit (CPU) intensive, especially for complex circuits such as high-speed VLSI interconnects [1]. In addition, circuit analysis and optimization is an iterative process in nature and requires repeated circuit simulation, which is CPU time consuming. To address this problem, the following two types of approximations have been previously utilized: the multidimensional polynomial (or its variants such as splines or response surface) models, (e.g., [2], [4], to approximate and replace original simulations during optimization; however, this approach can handle only mild nonlinearity in high-dimensional space. It typically requires model building or updating during optimization, consuming valuable on-line CPU time), and the lookup table approach, (e.g., [5], [6], to approximate and to replace accurate device or circuit simulations). However, the size of the table exponentially grows with dimension and the table becomes too difficult to generate and manage when many parameters of a device or a circuit are involved.

On the other hand, neural networks have been proposed to solve a wide variety of problems apart from the signal-processing area. It has been applied to microwave impedance matching [7] to study the effects of design factors on printed circuit-board (PCB) assembly yield [8], in modeling the properties of silicon–dioxide films [9], in via modeling [10] and minimization problems [11], in manufacturing process modeling [12], and in monolithic-microwave integrated-circuit (MMIC) passive-element modeling [13]. Recently, the feasibility and efficiency of using neural networks for physics-based device and circuit-level modeling for interactive computer-aided design (CAD) and optimization have been demonstrated [14]–[16]. The approach uses a feed-forward neural-network model to represent devices and circuits. The model is much simpler than device physics equations while retaining similar accuracy. Much computation is shifted from on-line optimization to off-line training of neural-network models.

Backpropagation (BP) [17], [18] is probably the most common algorithm used today in training feed-forward multilayer perceptron (MLP) neural networks. However, BP despite its popularity, suffers from serious limitations such as the computation effort and longer training times. This is also true in our application of circuit optimization and statistical design [15]. To alleviate these difficulties, much studying about algorithms has been done to achieve faster learning. These algorithms are based either on changing the network structure [20], using new activation functions [21], or turning to optimization techniques [22]–[24].

In this paper, we present a novel approach for fast training of feed-forward MLP neural networks based on the sparse matrix and decomposition concepts. The sparsity concept is commonly used in the circuit-analysis area, but its potential has not been fully exploited in the neural-network area. The new training technique, which we call the sparse-training technique, does not require any structure change in neural networks, nor does it require an initial network that is larger than necessary, as is the case for most pruning techniques [25]–[27], but instead makes use of the inherent nature of neural networks

Manuscript received October 1, 1996; revised June 20, 1997. This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

A. H. Zaabab is with the Department of Electronics, Carleton University, Ottawa, Ont., K1S 5B6 Canada. He is also with Semiconductor Insights Inc., Kanata, Ont., K2K 2A9 Canada.

Q.-J. Zhang and M. S. Nakhla are with the Department of Electronics, Carleton University, Ottawa, Ont., K1S 5B6 Canada.

Publisher Item Identifier S 0018-9480(97)07104-4.

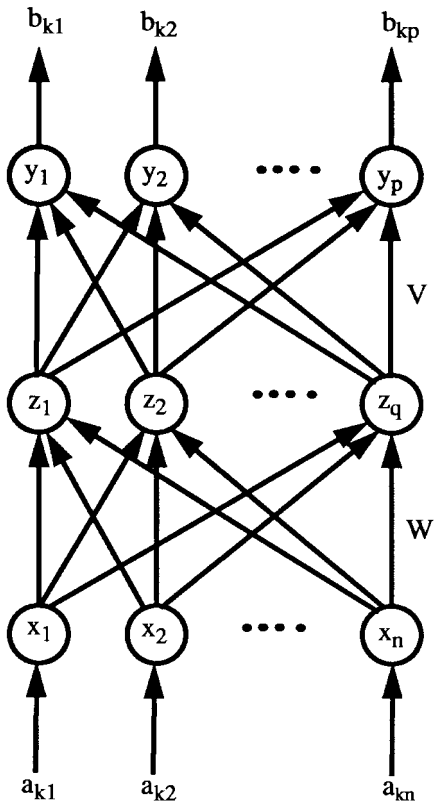


Fig. 1. Standard neural-network configuration.

that for each sample (some neuron activations are close to zero), gives rise to sparsity in the network, and hence, have no effect on network outputs and weights update. Much of the computation effort is saved while achieving the same accuracy if on a pattern basis these neurons are not taken into consideration in the learning process. It is understood that the sparse approach is applied once the neural-network structure is optimized, i.e., the best suitable number of hidden neurons is determined.

The sparse technique is used to train neural networks for both device- and circuit-level modeling. At the device level, the neural network represents a physics-oriented FET model, yet without the need to solve device physics equations repeatedly during optimization. At the circuit level, the neural network speeds up optimization by replacing repeated circuit simulations. Compared to standard BP training (hereafter referred to as the *dense technique*) the sparse approach yields CPU speed up while maintaining the same accuracy.

II. Neural-Network Models for Devices and Circuits

A neural network consists of a collection of interconnected neurons. Let $\mathbf{x}$ be a n-vector containing parameters of a given device or a circuit, e.g., gate-length and gatewidth of a FET or geometrical and physical parameters of high-speed VLSI interconnects [1], etc. Let $\mathbf{y}$ be a p-vector representing various responses of the device or the circuit under consideration, e.g., drain current of an FET. The relationship between $\mathbf{x}$ and response $\mathbf{y}$ is multidimensional and nonlinear.

A. Neural-Network Models

To model such a nonlinear relationship, a multilayer neural network is employed. We use a three-layer neural network with n neurons in the input layer, p neurons in the output layer, and q neurons in the hidden layer, as shown in Fig. 1. The input and output layers correspond to device or circuit parameters $\mathbf{x}$ and output responses $\mathbf{y}$, respectively. The hidden layer is represented by a q-vector $\mathbf{z}$. Let $\mathbf{a}_k = [a_{k1}\ a_{k2}\ \cdots\ a_{kn}]^T$ and $\mathbf{b}_k = [b_{k1}\ b_{k2}\ \cdots\ b_{kp}]^T$ be vectors representing the kth sample s_k of the inputs and outputs, respectively, $k = 1, 2, \cdots, N$, where N is the total number of data samples. The weighting factors between the input and the hidden layers are w_{ih}, and between the hidden and the output layers are v_{hj}, where $i = 1, 2, \cdots, n$, $h = 1, 2, \cdots, q$, and $j = 1, 2, \cdots, p$. The output from the neural network can be computed from circuit and device parameters x_i as

$$y_j = \sum_{h=1}^{q} z_h v_{hj} \tag{1}$$

where z_h is the output activation of the hth hidden neuron, defined as

$$\begin{aligned} z_h &= \frac{1}{1+\exp\left[-\left(\sum_{i=1}^{n} a_{ki} w_{ih}\right) - \theta_h\right]} \\ &= \frac{1}{1+\exp\left[-\left(\sum_{i=1}^{n} x_{ki} w_{ih}\right) - \theta_h\right]} \end{aligned} \tag{2}$$

and where θ_h is a threshold value for the hth hidden neuron.

B. Model Determination with Conventional Dense-Training Approach

The procedure to determine parameters in the neural-network model, is called neural-network learning or training. During learning, the neural network automatically adjusts its weights and thresholds (i.e., w_{ih}, v_{hj}, and θ_h) so that the error E between neural-network predicted y_j and sampled outputs b_{kj} is as follows:

$$E = \sum_{k=1}^{N} E^k = \sum_{k=1}^{N} \left[\frac{1}{2}\sum_{j=1}^{p} (y_j - b_{kj})^2\right] \tag{3}$$

and is minimized. The weights and thresholds update equations are given in [15]. In a conventional dense-training technique, all weights and all thresholds are updated for each sample, and this requires computing the error gradient for each weight and for each threshold in the network.

The sample data $s_k = (\mathbf{a}_k, \mathbf{b}_k)$ can be obtained by device or circuit simulations done off-line, or obtained directly from measurement. Equations (1) and (2) constitute the forward propagation, while (3) and the update equations constitute the backward propagation. During training, an epoch represents one forward and backward propagation through all training samples. Many epochs are needed to minimize the error E for a complete training process. The model parameters are then the final set of values w_{ih}, v_{hj}, and θ_h.

III. A Sparse-Training Approach to Model Determination

It is an inherent nature of neural-network models that on a sample pattern basis, hidden neurons have different activations, i.e., some neurons are zero or close to zero and some others are not. Our approach makes use of this inherent information to speed up the training process as follows.

Consider the same three-layer neural network described in Section II. Let H be the set containing all hidden neurons indexes, i.e., $H = \{1, 2, \cdots, q\}$. Let $S_{\rm th}$ be a small positive number representing a threshold value. In the neural-network model structure, the outputs y_j and all error gradients $\partial E^k/\partial v_{hj}$, $\partial E^k/\partial w_{ih}$, and $\partial E^k/\partial \theta_h$ are proportional to z_h. If, for a sample pattern s_k, the output of a hidden neuron $z_h \leq S_{\rm th}$, then that neuron has no effect on the network outputs and weights for that particular sample pattern. We call $S_{\rm th}$ the sparsity threshold. The sparse matrix concept is to associate each sample s_k with a sparse-index set H_k containing a subset of hidden neurons that have an effect on the network and disregard the others. In other words

$$H_k = \{h | z_h \geq S_{\rm th}, h = 1, 2, \cdots, q\}, \quad \text{where } k = 1, 2, \cdots, N \tag{4}$$

where q is the total number of hidden neurons and N is the total number of samples. We then have $H_k \subset H$. The sparse-index sets H_k, $k = 1, 2, \cdots, N$, are not necessarily disjoint. Fig. 2 best illustrates the sparsity concept. Once the subsets H_k are determined, the computation effort in the forward and backward propagations is substantially reduced by using the sparse technique as

$$y_j = \sum_{h \in H_k} z_h v_{hj} \tag{5}$$

$$z_h = \frac{1}{1 + \exp\left[-\left(\sum_{i=1}^{n} a_{ki} w_{ih}\right) - \theta_h\right]} = \frac{1}{1 + \exp\left[-\left(\sum_{i=1}^{n} x_{ki} w_{ih}\right) - \theta_h\right]}, \qquad h \in H_k \tag{6}$$

where

$$v_{hj}^{k+1} = v_{hj}^{k} - \eta \frac{\partial E^k}{\partial v_{hj}} + \alpha(v_{hj}^{k} - v_{hj}^{k-1}), \qquad h \in H_k \tag{7}$$

$$w_{ih}^{k+1} = w_{ih}^{k} - \eta \frac{\partial E^k}{\partial w_{ih}} + \alpha(w_{ih}^{k} - w_{ih}^{k-1}), \qquad h \in H_k \tag{8}$$

and

$$\theta_{h}^{k+1} = \theta_{h}^{k} - \eta \frac{\partial E^k}{\partial \theta_{h}} + \alpha(\theta_{h}^{k} - \theta_{h}^{k-1}), \qquad h \in H_k \tag{9}$$

where η and α are positive-valued learning rate and momentum, respectively. The sensitivity through the neural network

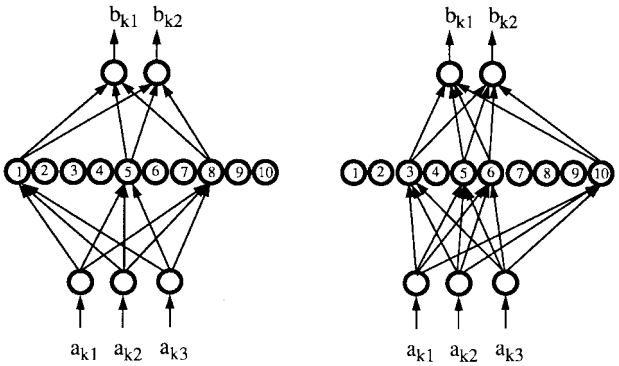


Fig. 2. Example of a neural-network configuration showing sparsity and sparse-index sets.

using the sparse technique is computed as

$$\frac{\partial E^k}{\partial v_{hj}} = (y_j - b_{kj}) z_h, \qquad h \in H_k \tag{10}$$

$$\frac{\partial E^k}{\partial w_{ih}} = z_h(1 - z_h) \sum_{j=1}^{p} (y_j - b_{kj}) v_{hj} a_{ki}, \qquad h \in H_k \tag{11}$$

and

$$\frac{\partial E^k}{\partial \theta_h} = z_h(1 - z_h) \sum_{j=1}^{p} (y_j - b_{kj}) v_{hj}, \qquad h \in H_k. \tag{12}$$

In Fig. 3, the average percentage sparsity of the network is plotted against the number of epochs. The change of sparsity, as depicted clearly in Fig. 3, is due to the fact that each hidden neuron becomes active one time or the other for at least one or more presented samples. This implies that all weights and biases are adjusted between epochs, and hence, hidden neuron activations could change and subsequently alter the network sparsity. It is then necessary to refresh the sparse-index sets H_k frequently during training. A refresh-cycle parameter R is consequently introduced and should be chosen so that the training time is minimized while not compromising the accuracy.

IV. Sparse-Training Algorithm

A. Algorithm

The sparse-training technique is built in conjunction with the BP learning algorithm and begins with finding the sparse-index sets H_k. Training then proceeds with processing only hidden nodes belonging to H_k while avoiding the others. Learning-rate adaptation and the use of a momentum term which are useful to further speed up the training process and avoid settling in an early local minimum could also be used in addition to the sparse-learning algorithm as follows.

Step 1. Choose the number of hidden neurons q and initialize the weights w_{ih}, v_{hj}, and thresholds θ_h with

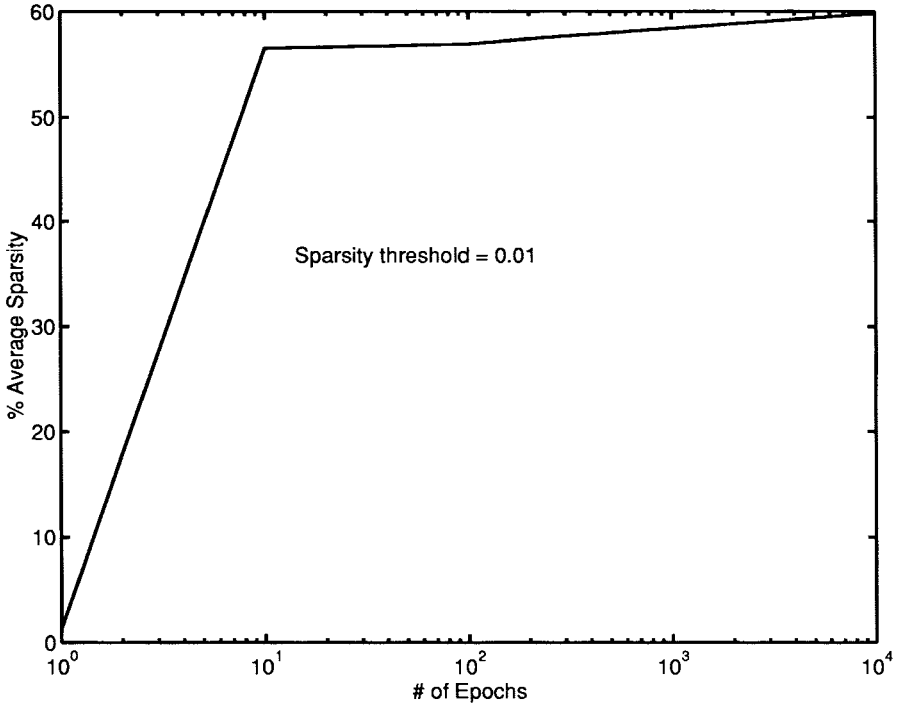


Fig. 3. Trend of network sparsity as training proceeds. The average sparsity is calculated according to (13).

small random numbers. Choose initial values for η and α.

Step 2. Choose values for the sparsity threshold $S_{\rm th}$, refresh-cycle R, and number of iterations N_E. Set $r = 1$, $n_E = 1$.

Step 3. Sparsity index set refresh cycle:

- For each sample s_k, calculate only the output of all hidden neurons and determine the sparsity index set H_k following (4).

Step 4. Set $k = 1$.

Step 5. Supply training sample $s_k = (\mathbf{a}_k, \mathbf{b}_k)$, let $\mathbf{x} = \mathbf{a}_k$.

Step 6. Forward propagation:

- Compute the network's output y following (5) and (6), considering only hidden neurons $h \in H_k$.

Step 7. Back propagation of the error:

- Compute the error E^k given in (3), and the gradients $\partial E^k/\partial v_{hj}$, $\partial E^k/\partial w_{ih}$, and $\partial E^k/\partial \theta_h$ in (10)–(12), where $h \in H_k$.
- Adjust the parameters of the network w_{ih}, v_{hj}, and θ_h using (7)–(9) with $h \in H_k$.

Step 8. $k = k + 1$, if $k \leq N$ go to Step 5.

Step 9. Compute the cumulative error E. If the cumulative error E is less than a given training tolerance ϵ, stop the training process.

Step 10. Adapt learning rate η and momentum α.

Step 11. $r = r + 1$, if $r \leq R$ then go to Step 4.

Step 12. $n_E = n_E + 1$, if $n_E \leq N_E$ go to Step 3 to refresh the sparse-index sets.

Step 13. Stop.

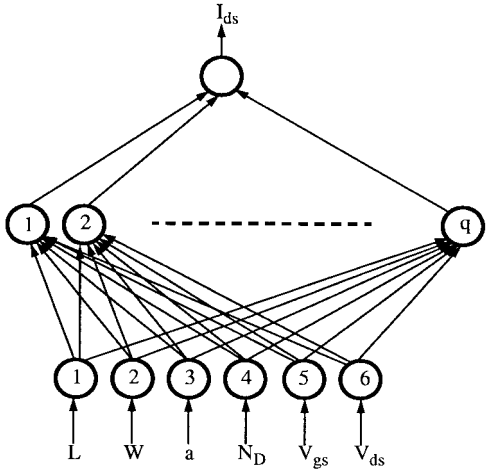


Fig. 4. Neural network for the MESFET model.

B. Sparse-Training Parameters

The efficiency of the sparse-training technique depends on the following parameters.

- Sparsity threshold $S_{\rm th}$: This parameter controls the sparsity threshold level of the network, so it should be chosen carefully. A too large value for this parameter implies faster training but may affect the network accuracy and generalization.
- Sparsity refresh-cycle R: This parameter represents the number of elapsed training epochs after which the sparse-index sets H_k are recalculated or refreshed. Since generally the network sparsity change is substantial at the beginning of training while it stabilizes around a constant value at the end of training as shown in Fig. 3, this parameter should be made small at the start and gradually

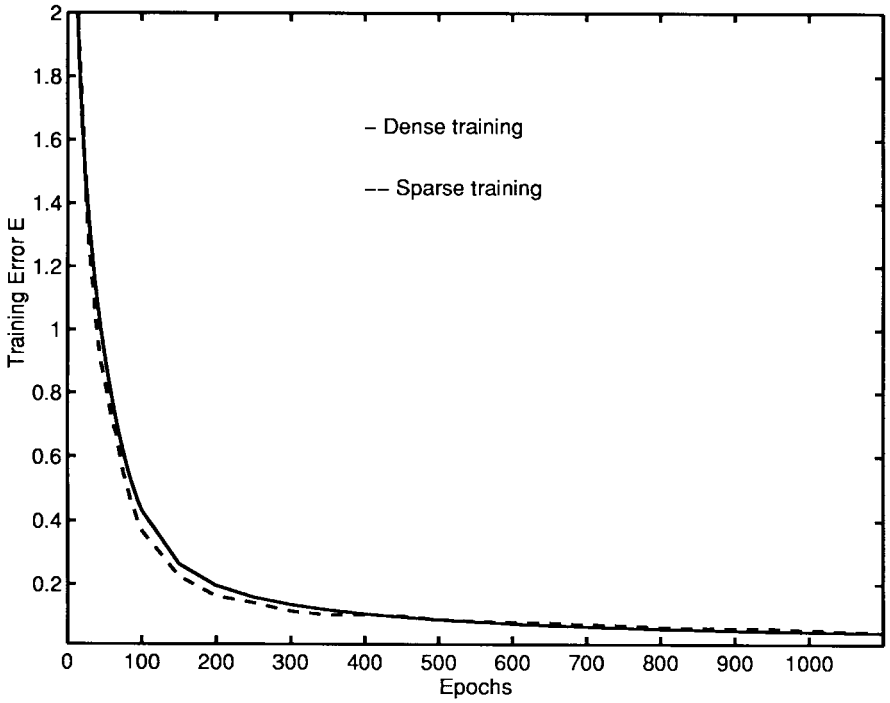


Fig. 5. Network learning curve for both dense- and sparse-training techniques. The training error E is computed following (3).

increased toward the end of training.

- Learning rate η and momentum α: The benefit as well as the choice of these parameters remain the same as for the dense technique [15].

The optimal values of the above-mentioned parameters are problem-dependent. Actual values of all these training parameters are given in examples 1 and 2 in Section V.

C. Estimation of Speed-Up Ratio

This applies for the weight adjustments in the backward pass. Let 1) q_k be the number of hidden nodes in each sparse-index set H_k, $k = 1, 2, \cdots, N$; 2) T be the number of total epochs used for training; and 3) Θ be the computation effort per sample required to update all the weights and biases linked to one hidden neuron, i.e., V_{hj}, W_{ih}, and θ_h, $j = 1, 2, \cdots, p$, $i = 1, 2, \cdots, n$, and h is fixed.

For the standard approach, all q hidden neurons are used for each of the N samples. For the sparse approach, assuming that the sparse pattern does not change between epochs, the network average sparsity is

$$\text{Average sparsity} = 1 - \frac{\sum_{k=1}^{N} q_k}{qN} \tag{13}$$

and the speed-up ratio is then

$$\text{Speed-up ratio} = \frac{T\Theta qN}{T\Theta \sum_{k=1}^{N} q_k} = \frac{qN}{\sum_{k=1}^{N} q_k} \tag{14}$$

since $q_k \leq q$, the above speed-up ratio is always greater than one.

TABLE I
SUMMARY OF SPARSITY RESULTS, ON SPARCSTATION 20. FET EXAMPLE: $q = 100$, $N = 1000$. VALUES BASED ON 100 EPOCHS

S_{th}	R (Epochs)	Average Sparsity (%)	CPU (sec)	Savings in CPU(%)
0	—	0	122.35	—
0.001	10	31.1	90.60	26.0
0.01	10	56.9	63.25	48.3
0	—	0	116.59	—
0.001	20	31.6	82.7	29.0
0.01	20	57.4	53.45	54.2

V. EXAMPLES

A. Physics-Oriented Neural-Network Model of a MESFET Device

Physics-based device models are very CPU intensive, especially when used for optimization or iterative simulations. A neural-network model for this kind of device will be very efficient in speeding up simulation and optimization [15]. The physical FET model chosen is the Khatibzadeh and Trew model [28]. Fig. 4 shows the representation of the neural-network model input–output parameters, where $I_{\rm ds}$ is the drain-to-source conduction current.

A three-layer feed-forward neural network is used to model this FET. The input vector $\mathbf{x}$ for the neural network has six parameters (including physical parameters): gate-length L, gatewidth W, channel-thickness a, and doping-density $N_{\rm d}$, and the gate–source and drain–source voltages $V_{\rm gs}$ and $V_{\rm ds}$. The number of neurons q in the hidden layer is 100. To train the neural network, each input parameter is allowed to vary over a certain range, as used in [15]. Both standard

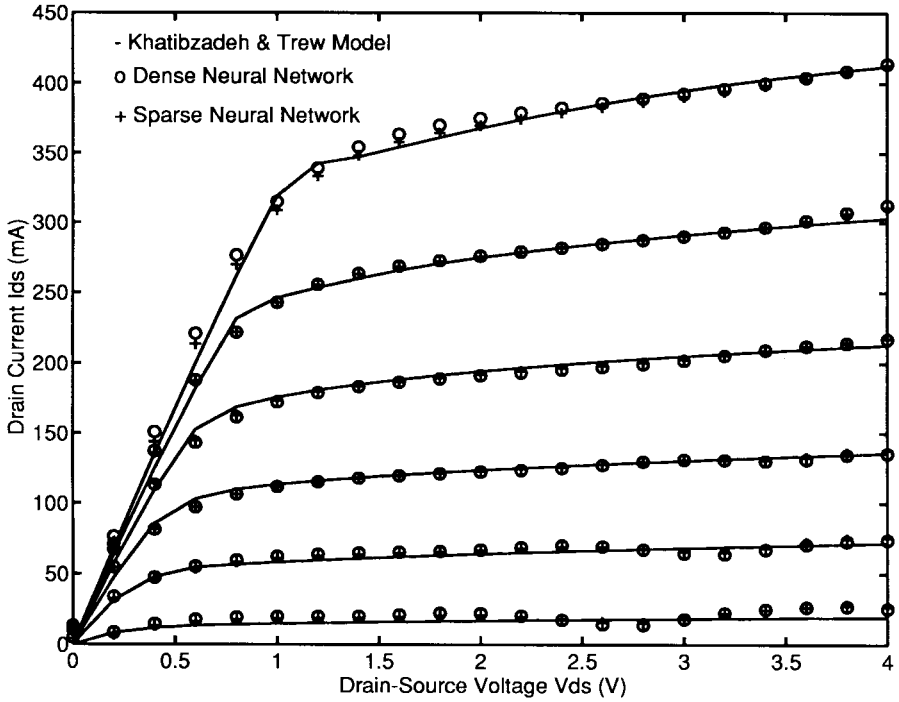


Fig. 6. Comparison of the dc characteristics using a neural-network model trained with the sparse technique (+) with that trained with the dense technique (o) and that of the Khatibzadeh and Trew model (−).

TABLE II
TREND OF SPARSITY WITH TRAINING

# Epochs	Average Sparsity %	
	$S_{th} = 0.001$	$S_{th} = 0.01$
0	0.02	1.3
10	30.6	56.5
100	31.1	56.9
200	31.6	57.4
10000	34.7	59.8

BP and proposed sparse-learning algorithms are used to train the network. Table I compares the CPU time results of both techniques after 100 epochs of training and for different values of sparsity threshold $S_{\rm th}$ and refresh-cycle R. Note that $S_{\rm th} = 0$ corresponds to the standard dense technique. On the other hand, Table II depicts the trend of network sparsity as training proceeds for different values of the sparsity-threshold parameter. Fig. 5 shows the network learning curve. The sparse-training technique may use similar or slightly more epochs compared to the dense technique, but since the computation per epoch is much less, the total training time achieved by the sparse approach is much lower.

We use new data which is different from the training samples for verification of the neural-network models. DC analysis predicted with our trained neural-network models are compared to those simulated using the original Khatibzadeh and Trew model shown in Fig. 6.

B. VLSI Interconnect Delay Modeling

Signal delay through interconnect networks is an important criteria in high-speed VLSI system design [1]. Fig. 7 repre-

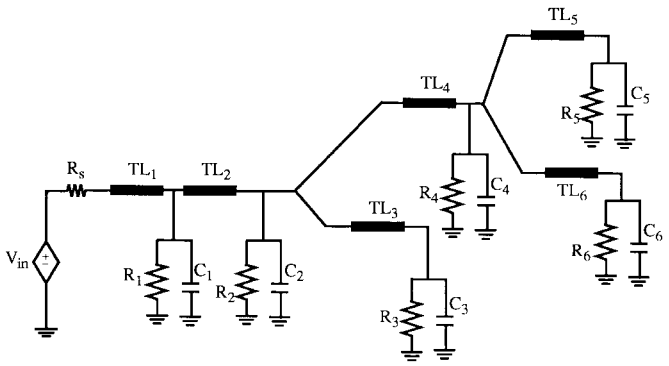


Fig. 7. Circuit model of a typical VLSI interconnect configuration in a PCB.

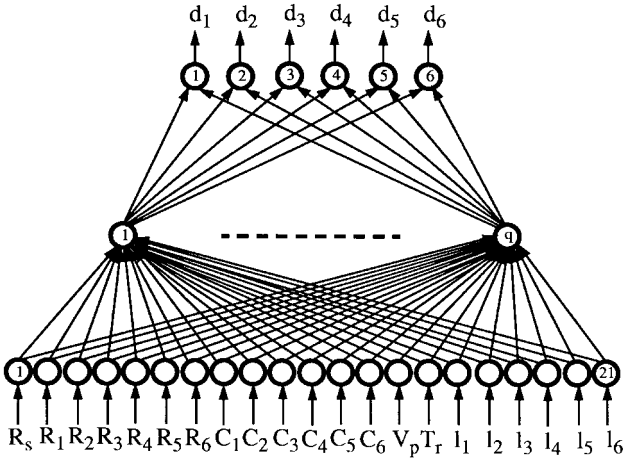


Fig. 8. Neural network for modeling the signal delay of the VLSI interconnect example.

TABLE III
RANGES OF NEURAL NETWORK INPUT PARAMETERS. VLSI-DELAY EXAMPLE

Parameters	Notation	Range	Unit	Type
Transmission line length	$l_i,\ i = 1 \cdots 6$	1 – 15	cm	Continuous
Termination Resistance	$R_i,\ i = 1 \cdots 6$	0.1, 3, 10, 20, 100	KΩ	Discrete
Termination Capacitance	$C_i,\ i = 1 \cdots 6$	3.3, 4.5, 5	pf	Discrete
Source Resistance	R_s	13.3, 20, 23.3, 40, 45	Ω	Discrete
Input Peak Voltage	V_p	0.8, 3.1, 3.2, 4.9, 5	V	Discrete
Input Rise Time	T_r	1.6, 2.5, 3.5, 5, 10	ns	Discrete

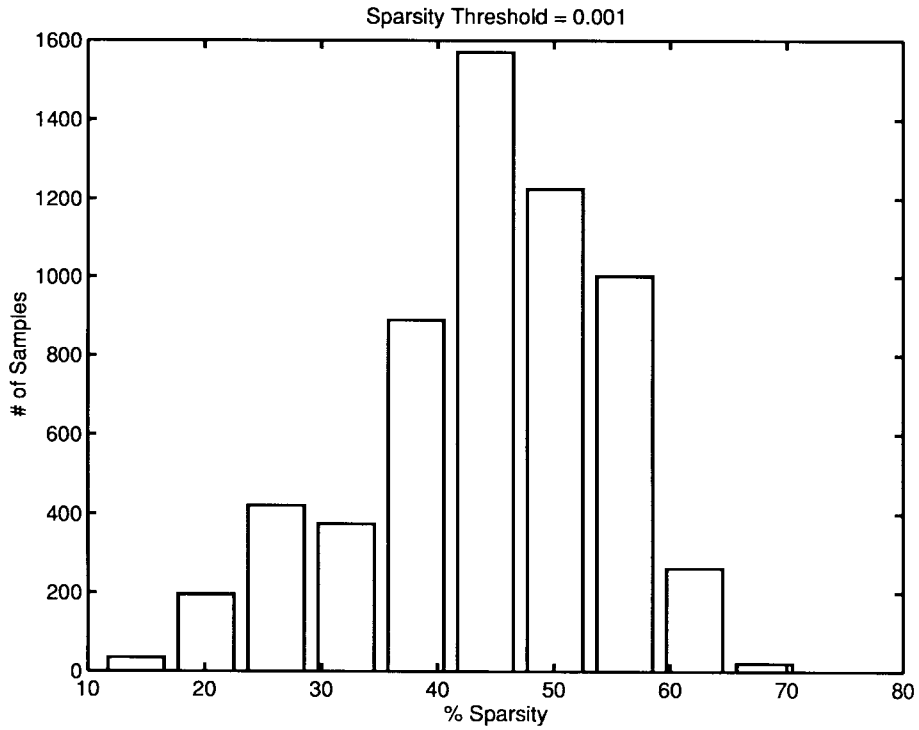


Fig. 9. Training samples' sparsity distribution for $S_{\rm th} = 0.001$.

sents a high-speed VLSI interconnect network modeled by six transmission lines and six *RC* termination ports. Repeated signal-delay analysis of this type of circuit is CPU intensive if done using conventional circuit simulators such as SPICE or Numerical Inverse Laplace Transform (NILT) [29], especially when a large number of parameters is considered.

A three-layer feed-forward neural network with 21 input nodes, 30 hidden nodes, and 6 output nodes is used, as shown in Fig. 8. The 21 input parameters are allowed to take on values so that all possible configurations which conform to standard industry requirements are considered. These ranges of values could be either discrete or continuous, as depicted in Table III. The six outputs are the propagation delays at the six termination ports, where delay is the time taken for the signal to reach 80% of its steady-state value. The sample data required to build this model was obtained by off-line simulation using NILT [29]. It is to be noted that the number of hidden nodes ($q = 30$) is the optimized value for this model, i.e., if the neural network is used with fewer hidden nodes, the model trained with a standard BP will not converge to a proper accuracy [16].

To clearly show how sparse the network is, Figs. 9 and 10 depict the distribution of sparsity over all 6000 training samples for a sparsity threshold $S_{\rm th} = 0.001$ and $S_{\rm th} = 0.01$, respectively. Table IV compares the CPU time used when utilizing both dense- and sparse-training techniques on a SPARCStation 5. After the network is trained, 4000 new configurations not used for training are used to test the new model. Fig. 11 shows the good agreement between the neural-network model trained with the sparse technique and original network simulation results. The CPU time used on a SPARCStation 2 to predict the delay of the 4000 configurations was 1.25 min for the neural-network approach while it was 3.4 h for the network simulation approach.

VI. CONCLUSIONS

In this paper, we have presented a novel neural-network training approach based on the sparse matrix concept. The developed technique uses the inherent property of neural networks in that the internal activations are usually sparse. Exploiting this feature, a new algorithm has been developed

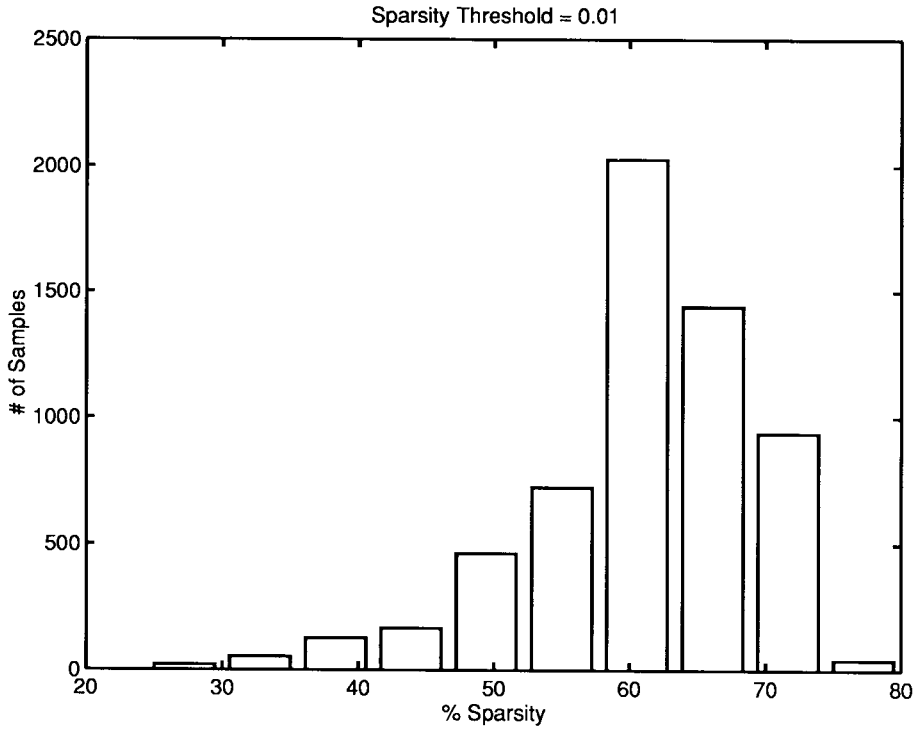
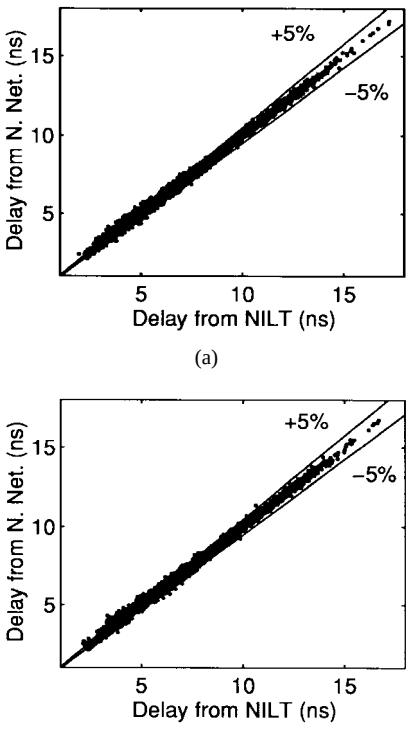


Fig. 10. Training samples' sparsity distribution for $S_{\rm th} = 0.01$.



(a)

(b)

Fig. 11. Comparison between signal delay obtained from the sparse neural-network model with that from original circuit simulation using NILT. (a) Signal delay at transmission line 5, d_5, and (b) delay at transmission line 6, d_6.

avoiding much unnecessary computations involved in the standard BP technique. Device-level and circuit-level neural-network models were generated using the sparse technique.

TABLE IV
SUMMARY OF SPARSITY RESULTS ON SPARCSTATION 5. INTERCONNECT EXAMPLE: HIDDEN NODES, 6000 SAMPLES. VALUES BASED ON 100 EPOCHS

S_{th}	R (Epochs)	Average Sparsity (%)	CPU (sec)	Savings in CPU(%)
0	10	0	899.24	—
0.001	10	46.37	563.22	37.4
0.01	10	63.82	444.21	50.6

The results demonstrated that much less CPU time was needed by the sparse technique to achieve the same model accuracy as that from the standard dense technique. The new technique results in a more efficient model development. The neural-network models developed can be used in an interactive CAD and optimization providing faster on-line solutions and speeding up design cycle.

REFERENCES

[1] M. Nakhla and Q. J. Zhang, Eds., *Modeling and Simulation of High-Speed VLSI Interconnects.* Norwell, MA: Kluwer, 1994.
[2] J. W. Bandler and S. H. Chen, "Circuit optimization: The state-of-the-art," *IEEE Trans. Microwave Theory Tech.*, vol. 36, pp. 424–443, Feb. 1988.
[3] J. W. Bandler, R. M. Biernacki, Q. Cai, S. H. Chen, S. Ye, and Q. J. Zhang, "Integrated physics-oriented statistical modeling, simulation and optimization," *IEEE Trans. Microwave Theory Tech.*, vol. 40, pp. 1374–1400, July 1992.
[4] R. M. Biernacki, J. W. Bandler, J. Song, and Q. J. Zhang, "Efficient quadratic approximation for statistical design," *IEEE Trans. Circuits Syst. I*, vol. 36, pp. 1449–1454, Nov. 1989.
[5] N. Jain, D. Agnew, and M. Nakhla, "Two-dimensional table lookup MOSFET model," in *Proc. IEEE Int. Conf. Comput.-Aided Design*, Santa Clara, CA, Sept. 1983, pp. 201–213.

[6] P. B. L. Meijer, "Fast and smooth highly nonlinear multidimensional table models for device modeling," *IEEE Trans. Circuits Syst. I*, vol. 37, pp. 335–346, Mar. 1990.
[7] M. Vai and S. Prasad, "Microwave circuit analysis and design by a massively distributed computing network," *IEEE Trans. Microwave Theory Tech.*, vol. 43, pp. 1087–1094, May 1995.
[8] Y. Li, R. L. Mahajan, and J. Tong, "Design factors and their effect on PCB assembly yield-statistical and neural network predictive models," *IEEE Trans. Comp., Packag., Manufact. Technol. A*, vol. 17, pp. 183–191, Feb. 1994.
[9] S. S. Han, S. A. Bidstrup, P. Kohl, and G. May, "Modeling the properties of PEECVD silicon dioxide films using optimized back-propagation neural networks," *IEEE Trans. Comp., Packag, Manufact. Technol. A*, vol. 17, pp. 174–182, Feb. 1994.
[10] P. M. Watson and K. C. Gupta, "EM-ANN models for microstrip vias and interconnects in dataset circuits," *IEEE Trans. Microwave Theory Tech.*, vol. 44, pp. 2495–2503, Dec. 1996.
[11] N. Funabiki and Y. Takefuji, "A neural network approach to topological via-minimization problems," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 770–779, June 1993.
[12] S. H. Huang and H. C. Zhang, "Artificial neural networks in manufacturing: Concepts, applications, and perspectives," *IEEE Trans. Comp., Packag., Manufact. Technol. A*, vol. 17, pp. 212–228, Feb. 1994.
[13] G. L. Creech, B. Paul, C. Lesniak, T. Jenkins, R. Lee, and M. Calcatera, "Artificial neural networks for accurate microwave CAD applications," in *IEEE Int. Microwave Symp. Digest*, San Francisco, CA, 1996, pp. 733–736.
[14] A. H. Zaabab, Q. J. Zhang, and M. Nakhla, "Analysis and optimization of microwave circuits and devices using neural-network models," in *IEEE Int. Microwave Symp. Digest*, San Diego, CA, 1994, pp. 393–396.
[15] ———, "A neural network modeling approach to circuit optimization and statistical design," *IEEE. Trans. Microwave Theory Tech.*, vol. 43, pp. 1349–1358, June 1995.
[16] ———, "Application of neural networks in circuit analysis," in *Proc. IEEE. Int. Conf. Neural Networks*, Perth, Australia, Nov. 1995, pp. 423–426.
[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation" in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1: Foundations, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, 1986.
[18] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE Acoustics, Speech, Signal Processing Mag.*, vol. 4, pp. 2–22, Apr. 1987.
[19] S. Haykin, *Neural Networks: A Comprehensive Foundation.* Piscataway, NJ: IEEE Press, 1994.
[20] M. Ishikawa, "Structural learning and its applications to rule extraction," in *Proc. IEEE Int. Conf. Neural Networks*, Orlando, FL, June 1994, pp. 354–359.
[21] B. W. Dahanayake and A. R. M. Upton, "A novel approach to fast learning: Smart neural nets," in *Proc. IEEE Int. Conf. Neural Networks*, Orlando, FL, June 1994, pp. 572–577.
[22] S. A. Barton, "A matrix method for optimizing a neural network," *Neural Computation*, vol. 3, pp. 450–459, 1991.
[23] S. Ergezinger and E. Thomsen, "An accelerated learning algorithm for multilayer perceptrons: Optimization layer by layer," *IEEE Trans. Neural Networks*, vol. 6, pp. 31–42, Jan. 1995.
[24] X. H. Yu, G. A. Chen, and S. X. Cheng, "Dynamic learning rate optimization of the backpropagation algorithm," *IEEE Trans. Neural Networks*, vol. 6, pp. 669–677, Jan. 1995.
[25] M. Ishikawa, "A structural learning algorithm with forgetting of link weights," Electrotechnical Lab., Tsukuba-City, Japan, Tech. Rep. TR-90-7, 1990.
[26] R. Reed, "Pruning algorithms—A survey," *IEEE Trans. Neural Networks*, vol. 4, pp. 740–747, Sept. 1993.
[27] Y. Le Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing (2)*, D. S. Touretzky, Ed. Denver, CO: Morgan Kauffmann, 1989, pp. 598–605.
[28] M. A. Khatibzadeh and R. J. Trew, "A large-signal, analytical model for the GaAs MESFET," *IEEE Trans. Microwave Theory Tech.*, vol. 36, pp. 231–238, Feb. 1988.
[29] R. Griffith and M. S. Nakhla, "Time-domain analysis of lossy coupled transmission lines," *IEEE Trans. Microwave Theory Tech.*, vol. 38, pp. 1480–1487, Oct. 1990.

A. Hafid Zaabab (S'91–M'96) received the Engineer Diploma in electrical engineering with honors from the National Institute of Electricity and Electronics (INELEC), Boumerdes, Algeria, in 1990, and the M.A.Sc. degree in electrical engineering from the University of Ottawa, Ottawa, Ont., Canada, in 1993. He is currently working toward the Ph.D. degree at Carleton University, Ottawa.

In 1993, he joined Carleton University, Electronics Department, as a Teaching Assistant and Research Engineer in VLSI circuit modeling and optimization. In 1995, he joined Semiconductor Insights Inc., IPS Department, Ottawa, where he focused on patent evaluation and integrated-circuit design analysis as they relate to intellectual property services, and in 1996, he joined the Design Analysis Department, where he is currently working on microcontrollers design issues. His research interests include neural-network modeling for circuit analysis and interactive CAD applications, high-speed VLSI CAD and optimization, and microcontrollers.

Mr. Zaabab was awarded a CIDA scholarship for the academic years 1991–1992 and 1992–1993.

Qi-Jun Zhang (S'84–M'85–SM'95) received the B.Eng. degree in electrical engineering from the East China Engineering Institute, Nanjing, China, in 1982, and the Ph.D. degree in electrical engineering from McMaster University, Hamilton, Ont., Canada, in 1987.

From 1982 to 1983, he was with the Institute of Systems Engineering, Tianjin University, Tianjin, China. From 1988 to 1990, he was a Research Engineer with Optimization Systems Associates Inc., Dundas, Ont., Canada. From 1989 to 1990, he was also a part-time Assistant Professor of electrical and computer engineering at McMaster University. In 1990, he joined the Department of Electronics, Carleton University, Ottawa, where he is currently an Associate Professor. He has co-edited *Modeling and Simulation of High-Speed VLSI Interconnects* (Kluwer, 1994), and has been a contributor to *Analog Methods for Computer-Aided Analysis and Diagnosis* (Marcel Dekker, 1988). His professional interests include all aspects of circuit CAD with emphasis on large-scale simulation and optimization, statistical design and modeling, optimization of microwave circuits and high-speed VLSI interconnections, neural entwork structures, training algorithms and applications to circuit design.

Michel S. Nakhla (S'83–M'85–SM'88) received the Ph.D. degree in electrical engineering from the University of Waterloo, Waterloo, Ont., in 1975.

From 1976 to 1988, he was with Bell–Northern Research (currently NORTEL Technology), Ottawa, Ont., Canada, as the Senior Manager of the computer-aided engineering group. In 1988, he joined Carleton University, Ottawa, where he is currently a Professor in the Department of Electronics and the Holder of the computer-aided engineering senior industrial chair established by Bell–Northern Research and the Natural Sciences and Engineering Research Council of Canada. He is founder of the high-speed CAD research group at Carleton University, and serves as technical consultant for several industrial organizations. He has authored or co-authored over 100 technical papers and two books on high-speed circuits and interconnects. His research interests include computer-aided design of VLSI and microwave circuits, high-frequency interconnects, synthesis of analog circuits, wavelets and neural networks.