INCREMENTAL COMMUNICATION FOR ARTIFICIAL NEURAL NETWORKS

by

Ali A. Ghorbani and Virendra C. Bhavsar

TR93-074 January 1993

Faculty of Computer Science
University of New Brunswick
P.O. Box 4400
Fredericton, N.B.
Canada E3B 5A3

Phone: (506) 453-4566
Fax: (506) 453-3566

INCREMENTAL COMMUNICATION FOR ARTIFICIAL NEURAL NETWORKS

Ali A. Ghorbani and Virendra C. Bhavsar
Parallel / Distributed Processing Group
Faculty of Computer Science
University of New Brunswick
Fredericton, N.B., E3B 5A3, Canada
Phone:+1(506)453-4566, Fax:+1(506)453-3566
Email: p3hs@unb.ca bhavsar@unb.ca

ABSTRACT

A learning procedure based on the backpropagation algorithm using the incremental communication is presented. In the incremental communication method instead of communicating the whole value of a variable, the increment or decrement to its previous value is only sent on a communication link. The incremental value may be either a fixed-point or a floating-point value. The method is applied to four different error backpropagation networks and the effect of the precision of the incremental values of activation, weights and error signals on the convergence behavior is examined. It is shown through simulation that at least 7-bit precision in fixed-point and 2-digit precision in floating-point representations are required for the network to generalize. With 12-bit fixed-point or 4-digit floating-point precision almost the same results are obtained as that with the conventional communication using 32-bit precision. The proposed method of communication can lead to enormous savings in the communication cost for implementations of artificial neural networks on parallel computers as well as direct hardware realizations. This method is applicable to many other types of artificial neural systems and can be incorporated along with the other limited precision strategies for representation of variables suggested in literature.

1. INTRODUCTION

The simulation of artificial neural networks is a compute intensive problem and even a moderate-sized network using the fastest scalar computers needs a large amount of computing time. The computing time can be drastically reduced by simulating neural networks on parallel computers and many such implementations have been carried out[4,20,22,23]. In practice, such an implementation involves more than mapping of neurons to processors. An efficient way of communication between processors that reduces the cost and increases the computational speed is required.

Full connectivity from external inputs to neurons or among neurons is a requirement in many artificial neural network models. Since the connectivity between different layers of nodes in neural networks is high, the intercommunication costs in parallel implementations are also very high. For example, for a feed-forward network with one single hidden layer consisting of $N_1 \times N_2 \times N_3$ nodes, where N_1 is the number of input nodes, N_2 is the number of hidden nodes and N_3 is the number of output nodes, the number of interconnections between nodes is $C = N_1 \times N_2 + N_2 \times N_3$. If $N_1 = N_2 = N_3$, the total number of interconnections is $O(N^2)$. Considering that the network requires more than one, say K, training sets to generalize, the overall intercommunication cost for an epoch turns out to be $O(KN^2)$; an epoch is one pass through the entire set of training examples. When M epochs are used for training, the total intercommunication cost is $O(KMN^2)$, which may be considered of fourth order.

With the intercommunication cost of order four and the fact that the output of a node usually depends on the output of many other nodes, it is obvious that the implementation of such networks on parallel computers can easily incur large communications cost. The main objective of this work is to reduce this intercommunication cost to allow faster simulation on parallel computers as well as to lower the cost for direct hardware implementations. One possibility is the use of less number of lines for the communication links (channels). However, in this case the full value data with full precision cannot be routed in one step. Therefore to

take advantage of narrower links we suggest the communication of incremental values with possibly reduced precision.

Two particular characteristics of the neural network models which contribute to the maximum theoretical parallelism are: (a) the high degree of connectivity among the neurons in the neural networks, and (b) the simultaneous activation of very large number of neurons. From an algorithmic point of view, this parallelism is extremely high and actually there is no computer at present that could support such parallelism for very large problem sizes.

Due to high degree of connectivity and large data flow in the operations of artificial neural networks, the structure and bandwidth of communication links to be used are of great importance. Presently most coarse/medium-grained systems are not able to provide sufficient interprocessor communication bandwidth. On the other hand, massively parallel fined-grained systems such as the CM2[4,20] seem to have insufficient local memory. To minimize communications time and cost in parallel implementations and overcome these inadequacies, we propose the use of incremental communication that requires smaller bandwidth for communication links. The proposed method of communication is applicable to many other types of artificial neural systems. It can also be used along with the other limited precision strategies for variables suggested in literature[2,8,10,17].

The outline of the paper is as follows. The concept of incremental communication is discussed in the following section. In Section 3, this method of communication is applied to the error backpropagation networks and a modified learning procedure based on standard error backpropagation algorithm using the incremental communication is presented. Subsequently, the effects of the incremental communication with different precision levels on the convergence behavior of four different networks are examined in Section 4. Finally, conclusions of the present study are summarized.

2. INCREMENTAL COMMUNICATION

In incremental intercommunication, instead of communicating the whole value of a variable, only the increment or decrement to its previous value is sent on a communication link. For example, assume that node Y requires to communicate the value of a variable X to node Z at different instants, as shown in Fig. 1(a). If $X(t)$ is the output of node Y at time t and $X(t+1)$ is its output at time $t+1$, in conventional communication the communication link will carry the value $X(t+1)$. In contrast, in the incremental communication the communication link will carry the value $\Delta X(t+1)$, where $\Delta X(t+1) = X(t+1) - X(t)$. Note that the new value $X(t+1)$ as well as the previous value $X(t)$ has to be stored at node Y. Moreover, we need to perform an extra operation (subtraction) to obtain the incremental value ΔX. At the receiving end, the value $X(t+1)$ will be obtained by updating the previous value $X(t)$ stored at Z with the $\Delta X(t+1)$, as given in Fig 1(b). Note that an extra operation (addition) is required at the receiving end. The idea of incremental communication has been employed earlier in delta communication and digital differential analyzers (DDAs)[1,14].

The number representation used for the incremental value ΔX may be fixed- or floating-point. The fixed-point value may be represented in the integer or fractional form using fewer numbers of bits (i.e., limited precision) than full-precision used for the variable X. In the floating-point representation, few bits of the mantissa and full value of the exponent are often used; this has been referred to as dynamic increment representation in DDAs[14]. The use of incremental exponent value is also possible. When the incremental value ΔX is limited to a smaller precision than the precision of X, we denote the incremental value by $\overline{\Delta X}$.

If the output of a node changes very fast, i.e. the slew rate for the variable X is high, then ΔX may not be small and several steps may be needed to communicate its value on a link having smaller number of lines. Alternatively, if the value of ΔX is restricted to a fewer number of bits (or digits), i.e. $\overline{\Delta X}$ is used instead of ΔX, for reducing the number of communication steps, the computational results may be affected. The cost of communication can be reduced by a large factor when ΔX is small most of the times and $\overline{\Delta X}$ can be passed to the next layer in

one step. In such a case, representing ΔX with limited precision will not have a major effect on computational results, because the amount of data that is lost by imposing restriction on the precision is not substantial.

The reduction factor in communication for sending a single value using the incremental communication over the conventional communication is given by

$$\omega = \alpha/\beta, \qquad ...(1)$$

where α represents the number of bits used to represent the full precision value X on the intercommunication link for the conventional communication and β represents the number of bits used to represent limited precision incremental value, $\overline{\Delta X}$, in the incremental communication.

Based on our preliminary experiments on several different backpropagation networks, we have found that when the learning coefficients are properly tuned, the magnitude of incremental values for the output of a node in most cases is smaller than the full-precision output values. This means that if limited precision incremental values of the output of a node are communicated instead of full precision output values, it would be more cost effective. The use of incremental values with limited precision reduces the intercommunications cost by a factor proportional to the precision used. The acceptable lower bound on the precision of incremental values is dependent on the type and complexity of the problem and learning algorithm as illustrated in Section 4.

The use of weights with limited precision and range has been suggested to reduce the cost of hardware implementations[2,10,17]. In this case full values of limited precision weights are communicated between neurons and no constraints are imposed on the other values involved in the neuron computations such as node's output values and error signals. The impact of imposing constraints on weight, sigmoid and weight-update has also been studied by few researchers[8,10]. Obviously, incremental communication can be incorporated in such implementations to further reduce the costs.

3. BACKPROPAGATION ALGORITHM

The backpropagation algorithm[9,13,19] is a supervised learning algorithm that trains a multilayer network by adjusting the link weights of the network using a set of " training examples". Each training example consists of an input pattern and an ideal output pattern that the user wants the network to produce for that input.

A typical backpropagation network is a feedforward network that has an input layer, an output layer and at least one hidden layer. There is no theoretical limit on the number of hidden layers, but typically there are one or two. The network topology is such that each node in a layer receives input from every node of the previous layer. Each node computes a weighted sum of all its inputs and then applies a nonlinear function on the weighted sum to yield the output value of the node. Sigmoid function, which is the most frequently used nonlinear activation function, is being used in this paper and in our simulation of incremental backpropagation.

The backpropagation learning algorithm can be represented as a set of matrix-matrix operations. The matrix notations can be easily exploited to represent two types of parallelism inherent in the backpropagation learning algorithm, namely intra-layer parallelism, i.e., parallel processing of many nodes of each layer of the network is performed, and training set parallelism, where multiple copies of the network are generated to process the training examples in parallel. In this paper some of the notations used for formulating the required equations for incremental backpropagation learning algorithm are from [20].

Let $\boldsymbol{I}$, $\boldsymbol{H}$, and $\boldsymbol{O}$ represent the matrices associated with the input, hidden and output layers, respectively. These matrices are of sizes $K \times N_1$, $K \times N_2$, and $K \times N_3$, respectively, where K is the total number of training patterns and N_1, N_2 and N_3 are the number of units (nodes) in input, hidden and output layers, respectively. The nonlinear transform $f(\ )$ is applied point-wise to the activity matrices to yield $\boldsymbol{I}^*$, $\boldsymbol{H}^*$ and $\boldsymbol{O}^*$. Let η represent the learning rate, $\otimes$ the point-wise multiplication operator and $\boldsymbol{D}$ the desired output matrix of size $K \times N_3$. Let δ_h represent the error signal matrix of size $K \times N_2$ for hidden layer and δ_o the error signal matrix

of size $K \times N_3$ for output layer. Let W_{ih} represent the connection weight matrix of size $N_1 \times N_2$ from input layer to hidden layer and W_{ho} represent the connection weight matrix of size $N_2 \times N_3$ for the weights from hidden layer to output layer. Finally, for all the above matrices let the same non-bold character as the name of the matrix represent an arbitrary element of the matrix; for example, H represents an arbitrary element of the matrix $\boldsymbol{H}$.

Based on the above notations, the equations in the backpropagation are given in Figure 2. Note that multiply-and-add is the basic operation in these calculations. The backpropagation algorithm consists of three steps:

1. Forward-pass, which computes the network output.
2. Backward-propagation, that computes the error at each node.
3. Weights-update, which adjusts the weights based on the errors.

These steps are briefly explained in the following subsections.

3.1. FORWARD PASS

In the forward-pass (recall) the computation does not need any data outside a node and the computation is local. The result of activation function, which is in most cases in the range of -1 to +1, is sent on the communication link to be used by the next node. Since the input vector is constant for the whole process, the computational cost of the first hidden layer during the forward passes can be reduced by storing the initial sum of weighted inputs and adding to it the sum of delta weighted inputs in each forward pass. Based on this we propose the following modified equations when incremental communication is used.

At the t-th learning step, the Equation (1) in Figure 2 gives

$$\boldsymbol{H}(t) = \boldsymbol{I}.\boldsymbol{W}_{ih}(t).$$

At the next $(t+1)$-th step

$$\begin{aligned} \boldsymbol{H}(t+1) &= \boldsymbol{I} \cdot \boldsymbol{W}_{ih}(t+1) \\ &= \boldsymbol{I} \cdot (\boldsymbol{W}_{ih}(t) + \Delta \boldsymbol{W}_{ih}(t)) \\ &= \boldsymbol{I} \cdot \boldsymbol{W}_{ih}(t) + \boldsymbol{I} \cdot \Delta \boldsymbol{W}_{ih}(t) \\ &= \boldsymbol{H}(t) + \Delta \boldsymbol{W}_{ih}(t) \cdot \boldsymbol{I} . \end{aligned}$$

Therefore, the increment to the value of $\boldsymbol{H}(t)$ to obtain $\boldsymbol{H}(t+1)$ is

$$\begin{aligned} \Delta \boldsymbol{H}(t+1) &= \boldsymbol{H}(t+1) - \boldsymbol{H}(t) \\ &= \Delta \boldsymbol{W}_{ih}(t) \cdot \boldsymbol{I}. \end{aligned} \qquad \ldots(3)$$

Also,

$$\Delta \boldsymbol{H}^{*}(t+1) = \boldsymbol{H}^{*}(t+1) - \boldsymbol{H}^{*}(t). \qquad \ldots(4)$$

If the magnitude of the increment ΔH^{*} is larger than that can be represented with the width, β, of the communication link, then it can be sent through many communication steps. Truncating the least significant bits of the increment ΔX to a variable X, and sending it with specified precision is an efficient way of reducing the cost of communication. The incremental value may be represented in the fixed-point or in the floating-point format; in the floating-point representation, fewer number of bits of the mantissa than in the standard communication are used. Let φ represent the function that converts ΔX to the limited precision fixed-point value $\overline{\Delta X}$ and ε represent the specified precision for fixed-point representation, then

$$\overline{\Delta X} = \varphi(\Delta X, \varepsilon). \qquad \ldots(5)$$

In the case of limited precision floating point representation, we use a function φ' to convert ΔX to $\overline{\Delta X}$ and ε' to represent the specified precision for floating-point communication. Thus for fixed-point incremental communication in backpropagation neural networks

$$\overline{\Delta \boldsymbol{H}}^{*}(t+1) = \varphi(\Delta \boldsymbol{H}^{*}(t+1), \varepsilon), \qquad \ldots(6)$$

and for floating-point incremental communication

$$\overline{\Delta H}^{*}(t+1) = \varphi'(\Delta H^{*}(t+1), \varepsilon'). \qquad ...(7)$$

Adapting ε and ε' as training progresses may be an efficient way of simulating artificial neural networks with limited precision incremental communication. In this case the required precision is determined based on the magnitude of the incremental value. Incremental values with larger magnitude employ larger number of bits whereas incremental values of smaller magnitude are represented with smaller precision.

3.2. BACKWARD PASS

In backward passes (training), to compute the error locally at each node in the hidden layer, we need to have both the incoming and outgoing weights of the node. This implies that we require communication with other nodes and therefore it is not entirely a local computation. Since there is no explicit target like desired output in the output layer for the hidden layer, to compute the error signals at the hidden layer we have to pass the weighted error signals from the bottom layer to the next upper layer. Thus,

$$\delta_h = f(I \cdot W_{ih}) \otimes (1 - f(I \cdot W_{ih})) \otimes (\delta_o^T \cdot W_{ho}). \qquad ...(8)$$

To reduce the load on communication links, we again propose the idea of incremental communication. Further, instead of passing the weighted error signals $\delta_o^T W_{ho}$ from the bottom layer (output layer) to next upper layer (hidden layer), we just pass the error signals δ_o^T. To implement the idea we have to have both the incoming and outgoing weights stored in each node, as required by the equation for δ_h given above. We can apply either incremental fixed-point (φ) or floating-point (φ') type conversion function to a δ, that is,

$$\overline{\delta} = \varphi(\delta, \varepsilon), \qquad \text{or} \qquad \overline{\delta} = \varphi'(\delta, \varepsilon') \qquad ...(9)$$

3.3. WEIGHT UPDATE

The weights are adjusted based on the difference between the ideal output and the actual output of the network. This can be seen as a gradient decent process in the weight space to minimize some error criteria. The connection weights can be updated after each training example, i.e., on-line update, or at the end of an epoch, i.e., batch update. In general an epoch is the number of sets of training data presented to the network (learning cycles) between weight updates. An overall error must be computed for the entire batch of training patterns. A more adequate error measure is the mean-square normalized error which is used in our simulation of backpropagation.

An extra momentum rate, γ, can be used in the equations (7) and (8) given in Fig. 2. This will accelerate the convergence time and prevent getting stuck at a local minima by adding a portion of the earlier change to the weight. Based on this the weights are adjusted according to :

$$\Delta W_{ih}(t+1) = \eta\ \delta_h^T \cdot I + \gamma\ \Delta W_{ih}(t) \qquad \Delta W_{ho}(t+1) = \eta\, \delta_o^T \cdot H + \gamma\ \Delta W_{ho}(t) \tag{10}$$

$$\overline{\Delta W}_{ih}(t+1) = \varphi(\Delta W_{ih}(t+1), \varepsilon) \qquad \overline{\Delta W}_{oh}(t+1) = \varphi(\Delta W_{oh}(t+1), \varepsilon). \tag{11}$$

Based on the type of updating strategy (i.e., batch or on-line) the time required to communicate incremental values in delta weight matrix, ΔW, is different. Since in batch update ΔW is communicated once in an epoch, its communication cost in batch update is much less than that in case of on-line update.

3.4. COMMUNICATION COST

Besides the reduction in the communication costs that is achieved by communicating the reduced precision values, the cost of communication can even further be reduced by computing delta weight vector, ΔW, of each node in a layer locally. In this case the current output value of each node is stored in the node to be used to compute the next incremental output value. After sending back the error signal, δ, from the bottom layer to the next upper

layer, a node in the upper layer has all the required data to compute the amount of adjustment to its outgoing weights. Therefore all nodes have the capability of adjusting both their incoming and outgoing weights and there is no need to send back delta weight vector from bottom layer to the next upper layer. This strategy accelerates the learning time and decreases the communication costs. Nevertheless there is always a trade-off between the cost of communicating delta weight vector, ΔW, between layers and the cost of repeating the computation of finding the outgoing delta weight vector, ΔW, in each node. Note that the nodes in the output layer do not have outgoing weights and as a result will not perform duplicate operations.

In the forward pass, incremental activation values with reduced precision, and in the backward pass, reduced precision error signals are communicated. Since the value of elements in error signal matrix, δ, is fairly small and the magnitude of incremental activation values is likely to be less than the magnitude of activation values, the communication cost will be reduced by incremental computation of node outputs. The possible reduction in the size of communication links is $\alpha - \beta$. For example, if the representation of each full precision activation value needs 32 bits and each activation incremental value in reduced precision needs 8 bits to be communicated, then the reduction in the cost of one link is (32-8) = 24 bits. Since the intercommunication cost is $O(n^4)$, the total reduction is $O(24^4)$ bits.

Each step in the backpropagation learning algorithm consists of two phases, namely recall and training phases. The overall intercommunication cost of this algorithm is of order $O(n^4)$ as stated in Section 1. In the case of on-line update where delta weight is communicated in each step the proportionality constant, c, in the complexity is at least equal to 3. When ΔW is computed locally and is not passed back between layers, the constant c is equal to 2 for both on-line and batch update strategies. Therefore based on the type of updating and communication strategies used, the actual reduction in terms of number of bits may be between $2 \cdot 24^4$ to $3 \cdot 24^4$ bits.

The saving in communication costs depends on the nature, complexity and the size of the problems. Initialization criteria and learning coefficients are also major factors. For example, the advantage to the continuous (on-line) update method is that the network starts learning examples immediately and usually finishes in fewer iterations. On the other hand, batch update has the advantage of saving both arithmetic and time. Apparently another advantage of doing batch update is that larger learning rates can be used without getting into local minimas. The use of larger learning rate will also result in smaller number of training cycles.

The reduction of communication cost in forward passes is achieved when the precision used for incremental values is considerably less than the precision used to represent the new output values. When β is considerably smaller than α and the size α is sufficient enough for the full precision output value to be communicated just in one step instead of multiple steps, the learning speed may be decreased and larger response time might result in certain cases. In the Section 4 we investigate these issues.

3.5. NODE ARCHITECTURE

A node is the basic processing element in the feedforward networks. The nodes in such networks can be classified as input, hidden, and output nodes. The architecture of a node using conventional communication is shown in Figure 3(a), whereas Figure 3(b) gives the architecture of a node using incremental communication. The learning process in both architectures is based on the on-line (continuous) update strategy. In these figures circle, solid and dotted rectangles represent an operation, a register and a temporary storage register, respectively. The rectangle with heavy lines represents memory with many words. The solid arrow represents the direction of flow of data.

The matrix notations used in the previous subsections are not sufficient to detail the components of the nodes. Therefore we introduce the following additional notations. Let L represent the number of layers in the network, N_ℓ represent the number of nodes in layer ℓ where $\ell = 1,2,3,...,L$ and n_ℓ represent an arbitrary node in layer ℓ, with $n_\ell = 1,2,3,...,N_\ell$.

Let $W_{n_\ell,n_{\ell+1}}^{\ell,\ell+1}$ represent a connection strength between node n_ℓ in layer ℓ and node $n_{\ell+1}$ in layer $\ell+1$. Let $net_{n_\ell}^{\ell}(t)$ and $A_{n_\ell}^{\ell}(t)$ represent the net-input and activation values of node n_ℓ in layer ℓ at step t, respectively. Finally let $D_{n_\ell}^{L,k}$ represent the n_ℓ-th pattern of desired output of k-th training example at the output layer (L-th layer).

The architecture in Fig. 3(a) is based on conventional communication method. It shows the basic components and principal operations of a hidden node. When it is used as an output node, the sum of weighted error signals, $\delta W^{\ell,\ell+1}$, is replaced with desired output. The desired output, D, is supplied by a teacher. The hidden node sends its output to the nodes in the next layer whereas the output node sends its output signal to the external world. The momentum is not incorporated in this architecture. A node in the backpropagation network with conventional communication performs the following operations.

In forward pass the node performs a multiply-and-add operation (inner product) to compute the sum of the weighted inputs (net-input value) and a non-linear activation function calculation to compute its output given by,

$$net_{n_\ell}^{\ell} = \sum_{n_{\ell-1}=0}^{N_{\ell-1}} W_{n_{\ell-1},n_\ell}^{\ell-1,\ell} A_{n_{\ell-1}}^{\ell-1} \qquad ...(12)$$

$$A_{n_\ell}^{\ell} = f(net_{n_\ell}^{\ell}). \qquad ...(13)$$

To carry out the required operations in the backward pass the node n_ℓ needs to keep some of the values from the forward pass. These values are : (a) the inputs that have been received by the node at the beginning of the forward pass, $A^{\ell-1}$, (b) its output value, $A_{n_\ell}^{\ell}$, that has been send to the nodes in the next upper layers (layer $\ell+1$), and (c) its incoming weights, $W^{\ell-1,\ell}$. These values are stored in the memories inside the node, as given in Fig. 3(a).

In the backward pass the node first receives the sum of the weighted error signals of those nodes to which it is connected. Subsequently, it applies the derivative of the nonlinear function (f') that is being used during the forward pass to yield the node's output value. Then, it computes its error signals by multiplying the input from forward pass to the result of

derivative function. Finally, the amount of adjustment to its incoming weights is computed and the weight-update operation is performed. These operations can be summarized as follows:

$$\delta_{n_\ell}^{\ell}(t) = f'(net_{n_\ell}^{\ell}(t)) \sum_{n_{\ell+1}=0}^{N_{\ell+1}} \delta_{n_{\ell+1}}^{\ell+1}(t) W_{n_\ell, n_{\ell+1}}^{\ell,\ell+1}(t) \qquad ...(14)$$

$$\Delta W_{n_{\ell-1}, n_\ell}^{\ell-1,\ell}(t) = \eta\, \delta_{n_\ell}^{\ell}(t) A_{n_{\ell-1}}^{\ell-1}(t) \qquad ...(15)$$

$$W_{n_{\ell-1}, n_\ell}^{\ell-1,\ell}(t+1) = W_{n_{\ell-1}, n_\ell}^{\ell-1,\ell}(t) + \Delta W_{n_{\ell-1}, n_\ell}^{\ell-1,\ell}(t). \qquad ...(16)$$

Figure 3(b) shows the architecture of a node in incremental backpropagation network. In forward pass, for every input, the node performs three additional operations compared to the conventional communication. These operations are: (a) an extra addition at the point of entry to the node X and prior to the multiply-and-sum operation in order to compute the new full-precision net-input value, (b) an additional subtraction at the point of exiting node X, after calculating the full-precision activation value, in order to compute the incremental output value of node X, and (c) a precision reduction operation, φ (or φ'), in order to reduce the precision of incremental output value.

Extra storage is needed to store previous input vector values and previous output value. Both incoming and outgoing weight vectors, $W^{\ell-1,\ell}$ and $W^{\ell,\ell+1}$, are also stored in the node. They are adjusted based on updating strategy, i.e., at the end of each training example in on-line update or at the end of each epoch in batch update. Although the architecture is shown for the on-line update strategy, with few minor changes it can easily be adapted for the batch update policy.

In the backward pass (training), based on the communication strategy (explained in Section 3.4), node n_ℓ either receives $\overline{\Delta} W_{n_\ell, n_{\ell+1}}^{\ell,\ell+1}$ from the upper layer (layer $\ell+1$) for updating the outgoing weights as

$$W_{n_\ell, n_{\ell+1}}^{\ell,\ell+1}(t+1) = W_{n_\ell, n_{\ell+1}}^{\ell,\ell+1}(t) + \overline{\Delta} W_{n_\ell, n_{\ell+1}}^{\ell,\ell+1}(t), \qquad ...(17)$$

or computes outgoing weight increment, $\Delta W_{n_\ell,n_{\ell+1}}^{\ell,\ell+1}$, locally by using the error signal vector, $\delta^{\ell+1}$, that is received from the upper layer (layer $\ell+1$),

$$W_{n_\ell,n_{\ell+1}}^{\ell,\ell+1}(t+1) = W_{n_\ell,n_{\ell+1}}^{\ell,\ell+1}(t) + \eta f(A_{n_\ell}^{\ell}(t)) \sum_{n_{\ell+1}=0}^{N_{\ell+1}} \delta_{n_{\ell+1}}^{\ell+1}(t) W_{n_\ell,n_{\ell+1}}^{\ell,\ell+1}(t), \qquad ...(18)$$

where

$$A_{n_\ell}^{\ell}(t) = f\Big(\sum_{n_{\ell-1}=0}^{N_{\ell-1}} W_{n_{\ell-1},n_\ell}^{\ell-1,\ell}(t) A_{n_{\ell-1}}^{\ell-1}(t)\Big). \qquad ...(19)$$

This architecture is much more suitable for cases where activation function uses the previous activation value $A_{n_\ell}^{\ell}(t-1)$ as well as the net-input value $net_{n_\ell}^{\ell}(t)$ (or $A_{n_{\ell-1}}^{\ell-1}(t)$) to compute the new output value $A_{n_\ell}^{\ell}(t)$, i.e.,

$$A_{n_\ell}^{\ell}(t) = f(A_{n_\ell}^{\ell}(t-1), net_{n_\ell}^{\ell}(t)). \qquad ...(20)$$

4. COMPUTATIONAL EXPERIMENTS

We have performed empirical studies to determine the effect of the precision of fixed-point and floating-point representation of incremental activation value, ΔA, error signal, δ, and delta weight, ΔW, on the convergence. In the simulation of error backpropagation networks we have implemented the batch update version of the algorithm on a SUN 4 workstation with one processor.

For the computational studies, the backpropagation learning parameters are tuned based on other researchers' experiments that are reported in the literature[5,6,11]. We assume that a network is converged if starting at a precise initial configuration after a number of epochs, it could learn to separate the input patterns according to the " *threshold and margin* " criterion. If the total range of the input units is 0.0 to 1.0, any value below 0.4 is considered to be zero and any value above 0.6 is considered to be a one; values between 0.4 and 0.6 are considered to be " *marginal* ", and are not counted as correct during training[6]. The experiments consist of training four different networks while varying the precision of incremental values of activation, error signal and delta weight in the fixed- or floating-point incremental communication.

Benchmark problems consist of four single hidden layer networks, all of them with $N_1 \times N_2 \times N_3$ architecture. These networks are well known and have been used in most experiments throughout the history of artificial neural networks[6]. In our investigation of incremental communication, these networks are used for measuring and comparing the effects of precision of the incremental values on the convergence. These networks are as follows:

- Network A: The XOR/parity network with one hidden unit and crosscut connections.
- Network B: The XOR/parity network with two hidden units and no crosscut connections.
- Network C: A classifier network with 4-4-1 architecture with 16 patterns in training set. The network classifies the inputs as 0 or 1.
- Network D: An encoder network with 64-20-1 architecture with 69 patterns in training set. The network classifies the inputs as 0 or 1.

Note that networks with crosscut connections are more communication intensive than networks without crosscut connections.

In the experiments for the fixed and floating-point incremental communication, the precision for fixed-point representation is varied from 7 to 12 bits in the steps of one bit and the precision of mantissa for floating-point representation is varied from 2 to 4 digits in the steps of one digit. The digit representation of mantissa in floating-point representation of incremental values is employed due to the ease of implementation. When the precision of 6-bit was used for delta weight, ΔW, and error signal, δ, the network were found failing to converge; the precision of 6-bit for incremental activation values was however acceptable for higher precision for ΔW and δ.

For all networks stated above we continue training until the normalized mean-square error of an epoch becomes less than 0.001 or for an entire training set every output is correct. The number of epochs is used to measure learning time and to compare the effectiveness of various precisions of fixed-point and floating-point incremental representations. The same precision is used for all incremental values in out experiment. In other words, the precisions of ΔA, ΔW and δ are kept the same during the course of learning. In this study all the

computations inside a node, viz. sigmoid, weight-update or error signal computation, are handled in full precision.

The result of experiments for Network A is shown in the Figures 4. Figure 4(a) represents the error for varying precision of fixed-point representation as a function of the number of epochs. It is seen that as the precision of incremental values increases, the number of epochs required for generalization gets closer to the number of epochs required for generalization with conventional communication. Figure 4(b) represents the error for various floating-point precision values for incremental communication and full floating-point precision values used for conventional communication. It is seen that the error exhibits properties similar to as given in Figure 4(a). A comparison of Figures 4(a) and 4(b) shows that, although both forms of communication generalize with different precision values, the learning time for the floating-point incremental representation is closer to the conventional learning time than that for the fixed-point incremental representation. This is expected since floating-point representation allows the representation of a larger range.

Figure 5 gives the errors for the application of the fixed- and floating-point incremental communications for Network B. With the use of fixed- and floating-point representations for the increments the network outputs converge for various precisions. Note that the learning time for the floating-point increment representation is much closer to that of the conventional communication, compared to the fixed-point incremental communication. The learning time with 4-digit mantissa in the floating-point representation is very similar to the conventional representation.

The results of applying conventional communication, fixed- and floating-point incremental communications to Network C are shown in Figure 6. Although there are some instabilities during the convergence, both the fixed- and floating-point representations converge for all the chosen precision values. Again the learning time for the floating-point incremental representation is closer to the conventional learning time than that for the fixed-point incremental representation.

Figure 7 shows the result of applying fixed- and floating-point incremental communications, respectively on Network D. Figure 7(a), which represents the results for the fixed-point incremental communication, shows instability in some cases. Both forms of incremental communication converge for various precisions. Figure 7(b) shows that the learning times of floating-point incremental communication for most of the precision values are very close to that of the conventional learning times.

The number of epochs required for generalization for the various networks using different incremental representations are summarized in Table 1. It is seen that in all cases the networks converge, but the convergence time gets closer to the conventional communication convergence time as the number of bits is increased. The average reduction in the communication cost of fixed-point incremental communication is more than that of floating-point incremental communication. On the contrary, the results for floating-point incremental communication are mostly better than those for the fixed-point incremental communication.

The incremental communication decreases the magnitudes of output value of various nodes in each training step by an amount proportional to $(\alpha - \beta)$. Based on the gradient descent algorithm, this may cause an increase/decrease in the required number of epochs for generalization. Communicating the limited precision incremental values means that all of the required information is not sent to the receiving node. Since partial data is sent to each node, it may again cause the network to generalize in a smaller/larger number of epochs.

In our other experiments on different networks we have found that 4-digit precision for incremental output values in floating-point communication and 12-bit precision in fixed-point communication are sufficient for the network to converge in the same time as conventional communication. For the case of ΔW and δ, 4-digit precision in floating-point communication and 13-bit precision in fixed-point communication is sufficient to produce the same result as for conventional communication.

5. CONCLUSION

A new method of communication for artificial neural networks is proposed. It is shown that the learning process in the neural networks is communication intensive. The proposed incremental inter-node communication architecture attempts to minimize the communication costs with a large factor by communicating only increments/decrements of variable values; the representation of incremental values of the activation, weights and error signals can be either fixed or floating-point. The communication costs can further be reduced by using limited precision for the incremental values. The precisions of various incremental values can be kept the same or may be different.

In our experiments with error backpropagation networks the increase in learning time (number of epochs) is found to be small. In fact, in some cases with an appropriate number of bits for incremental values, which is much smaller than the number of bits for the full-precision values, the learning time is very close to and sometimes even the same as the conventional communication.

The concept of incremental inter-node communication is applicable to large classes of artificial neural networks. It can also be used along with the other limited precision strategies for representing variables suggested in literature. The proposed method of communication can be applied for parallel implementation of the artificial neural networks on various multiprocessor architectures. The communication complexity of implementing incremental communication on multiprocessor computers with various topologies is discussed in our other report [7]. The incremental communication supports smaller bandwidth for the channels and therefore, decreases the required communication time. The incremental method and especially the fixed-point representation for incremental values is suitable for VLSI realization.

6. REFERENCES

[1] V.C. Bhavsar, "Special Purpose Computers, with emphasis on Digital Differential Analyzers, for Military Applications", M.Tech. Dissertation, India Institute of Technology, Bombay, India, 1973.

[2] D. D. Cavilia, M. Valle, and G. M. Bisio, "Effects of Weight Discretization on the Back Propagation Learning Method: Algorithm Design and Hardware Realization," In International Joint Conference on Neural Networks, San Diego, CA., Vol. 2, pp. 631-637, 1990.

[3] F. Distante, M. Sami, R. Stefanelli and G. Stori-Gajani, "Mapping Neural Nets onto a Massively Parallel Architecture: A Defect-Tolerance Solution," Proc. of the IEEE, Vol. 79, pp. 444-460, April 1991.

[4] E. Deprit, "Implementing Recurrent Back-propagation on the Connection Machine," Neural Networks, Vol. 2, pp. 295-307, 1990.

[5] H. A. C. Eaton and T. L. Olivier, " Learning Coefficient Dependence on Training Set Size," Neural Networks, Vol. 5, pp. 283-288, 1992.

[6] S. E. Fahlman, "An empirical study of learning speed in backpropagation networks," Rep. No. CMU-CS-88"-162, Carnegie Mellon Univ., Sept. 1988.

[7] A. A. Ghorbani and V. C. Bhavsar, "Communication Complexity of implementing artificial neural networks with incremental communication on multiprocessor computers," in preparation, 1993.

[8] M. Hoehfeld and S. E. Fahlman, "Learning with limited numerical precision using the cascade-correlation algorithm," IEEE Transaction on Neural Networks, Vol. 3, No. 4, pp. 602-611, July 1992.

[9] R. Hecht-Nielsen, "Theory of the backpropagation neural networks," In International Joint Conference on Neural Networks, Washington, DC., Vol. 1, pp. 593-605, 1989.

[10] P. W. Hollis, J. S. Harper, and J. J. Paulos, "The effects of precision constraints in a backpropagation learning network," Neural Computation 2, pp. 363-373, 1990.

[11] R. A. Jacobs, "Increased Rates of Convergence Through Learning Rate Adaptation," Neural Networks, Vol. 1, pp. 295-307, 1988.

[12] T. Lange, "Simulation of heterogeneous neural networks on serial and parallel machines," Parallel Computing , Vol. 14, pp. 287-303, 1990.

[13] R. P. Lippmann, "An introduction to computing with neural nets," IEEE Acoustics Speech Signal Process, Mag., Vol. 4, pp. 4-22, Apr. 1987.

[14] F.V. Mayorov, Electronic Digital Integration: Computers-Digital Differential Analyzers, Translated into English by Y. Chu, London: Iliffe Books Ltd., 1964.

[15] S. Makram-Ebeid, J. A. Sirat, and J. R. Vila, "A rationalized error back-propagation learning algorithm," In International Joint Conference on Neural Networks, Washington, DC., 1989.

[16] W. Ming, V.K.Prasanna and W.Przytula, "Algorithmic Mapping of Neural Network Models onto Parallel SIMD Machines," IEEE Transactions on Computers, Vol. 40, pp. 1390-1401, December 1991.

[17] K. Nakayama, S. Inomata, and Y. Takeuchi, "A digital multilayer neural network with limited binary expressions," In International Joint Conference on Neural Networks, San Diego, CA., Vol. 2, pp. 578-592, 1990.

[18] K. Obermayer, H. Ritter and K. Schulten, "Large-scale simulations of self-organizing neural networks on parallel computers: application to biological modeling," Parallel Computing, Vol. 14, pp. 381-404, 1990.

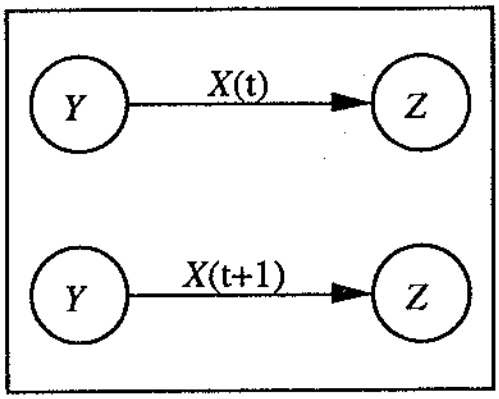
[19] D. E. Rumelhart, and J. L. McClelland, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vols. I and II, MIT Press, Cambridge, MA, 1986.

[20] A. Singer, "Implementations of artificial neural networks on Connection Machine," Parallel Computing, Vol. 14, pp. 305-315, 1990.

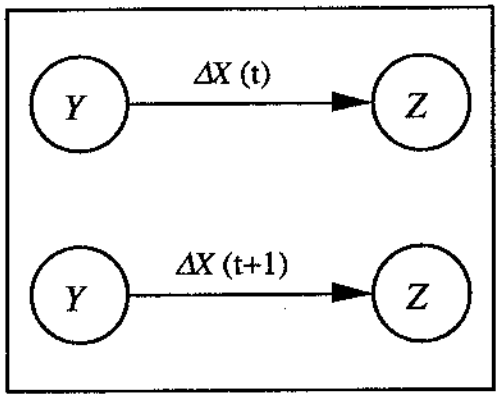
[21] M. Stevensson, R. Winter, and B. Widrow, "Sensitivity of feedforward neural networks to weight errors," IEEE Trans. Neural Networks 1(1), pp. 71-80, 1990.

[22] M. Witbrock and M. Zagha, "An implementation of backpropagation learning on GF11, a large SIMD parallel computer," Parallel Computing, Vol 14, pp. 329 346, 1990.

[23] Z. Zhang, M. Mckenna, J.P. Mesirov and L. Waltz, "The backpropagation algorithm on grid and hypercube architecture," Parallel Computing, Vol. 14, pp. 317-327, 1990.



a. Conventional Communication



$Y : \Delta X(t+1) = X(t+1) - X(t)$

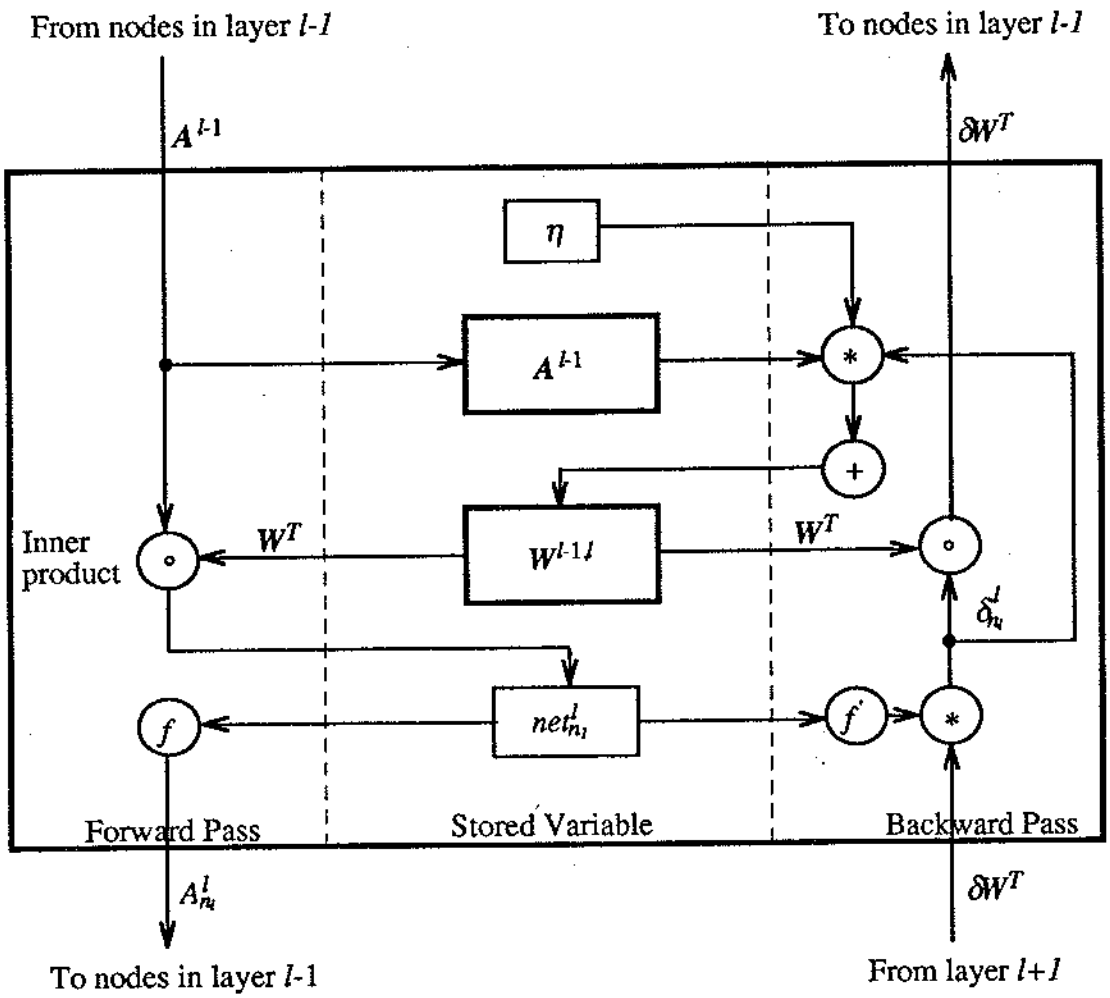
$Z : X(t+1) = X(t) + \Delta X(t+1)$

b. Incremental Communication

Figure 1. Conventional versus incremental communication

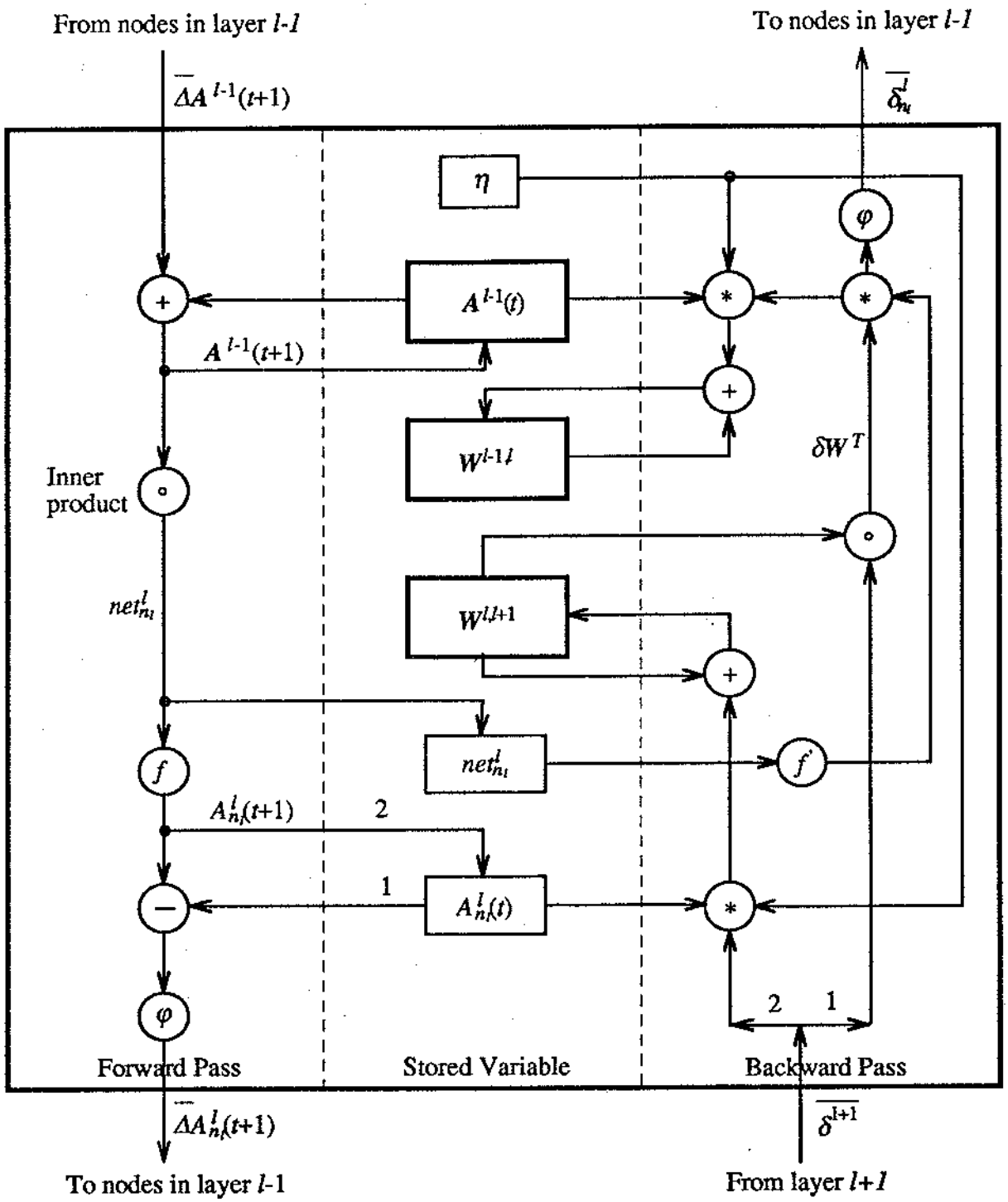
Forward Pass	Backward Pass
1) $H = I \cdot W_{ih}$	5) $\delta_o = O^* \otimes (1 - O^*) \otimes (D - O^*)$
2) $H^* = f(H)$	6) $\delta_h = H^* \otimes (1 - H^*) \otimes (\delta_o \cdot W_{ho}^T)$
3) $O = H^* \cdot W_{ho}$	7) $= W_{ho} = W_{ho} + \eta \delta_o^T \cdot H$
4) $O^* = f(O)$	8) $= W_{ih} = W_{ih} + \eta \delta_h^T \cdot I$

Figure 2. Backpropagation algorithm with conventional communication.



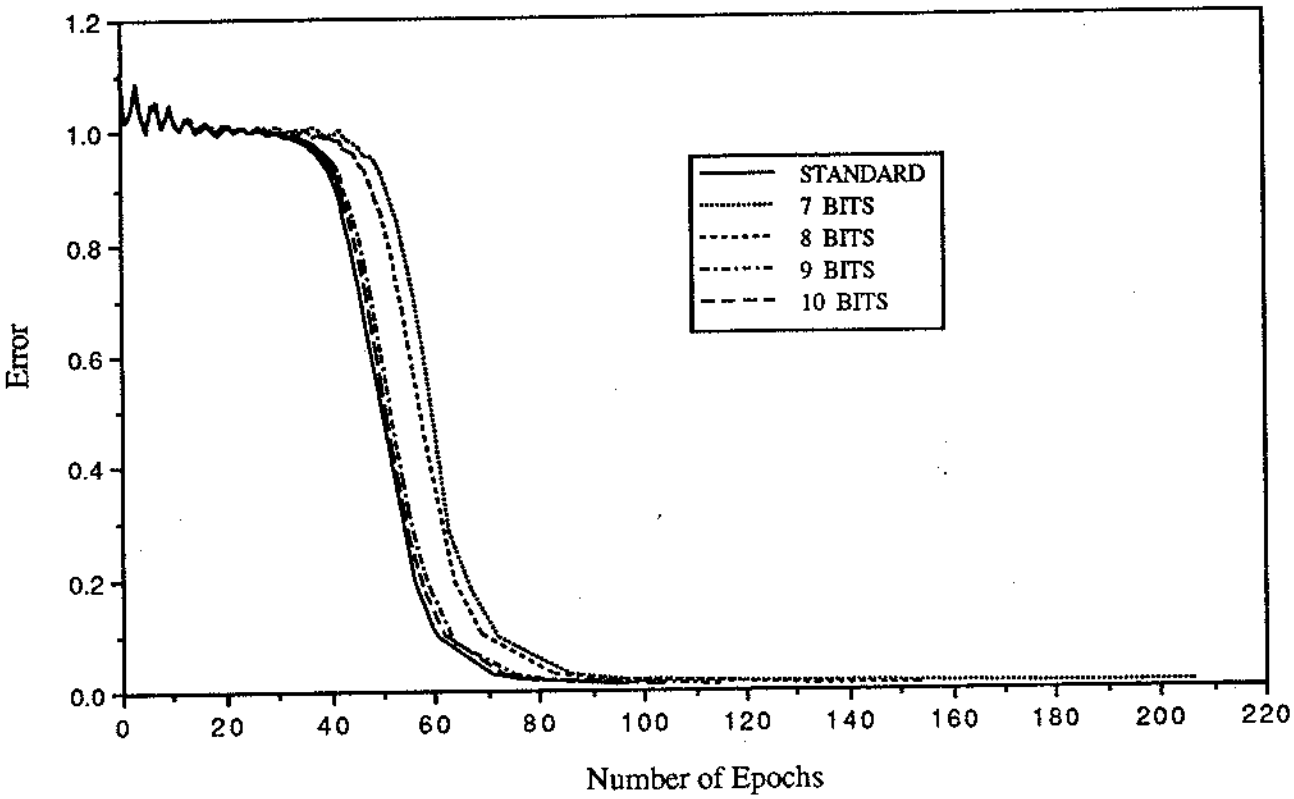
(a) Conventional Communication

Figure 3. Node architecture for on-line update.



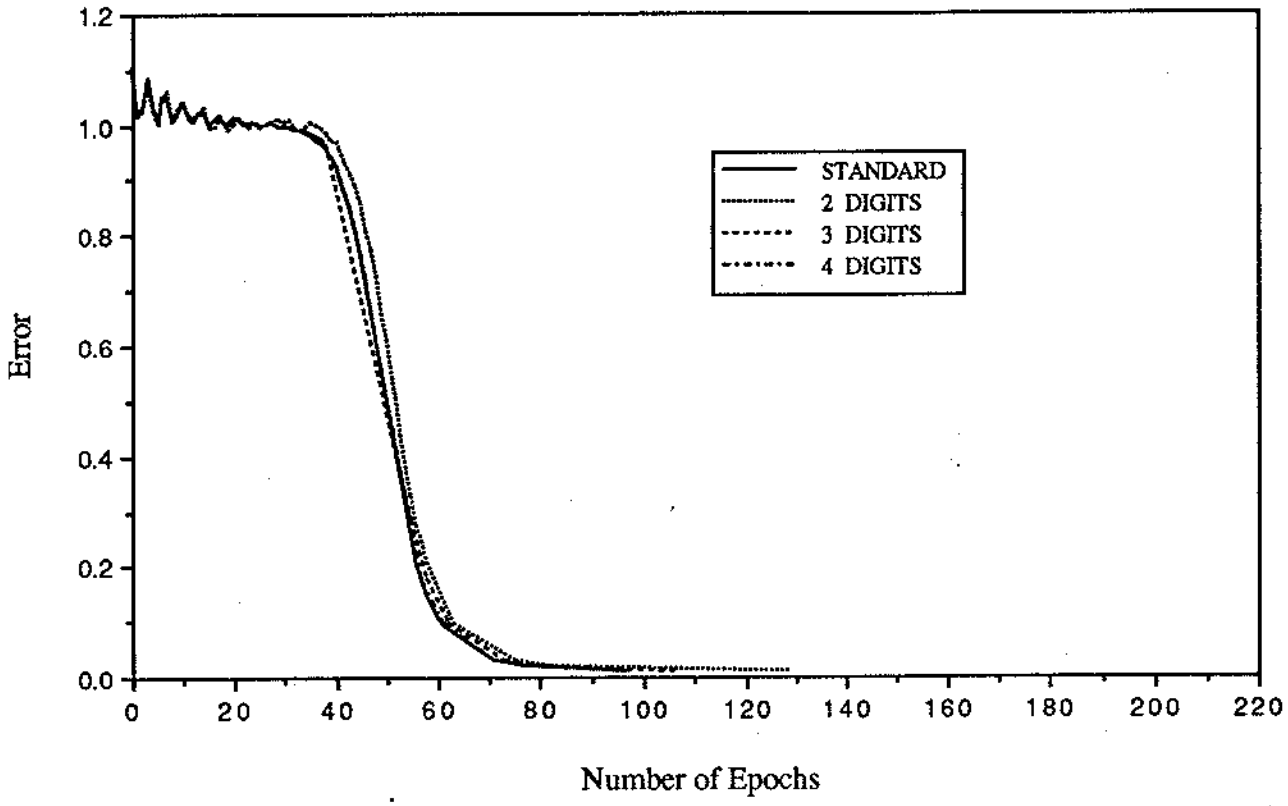
(b) Incremental Communication.

Figure 3. Node architecture for on-line update (cont.).



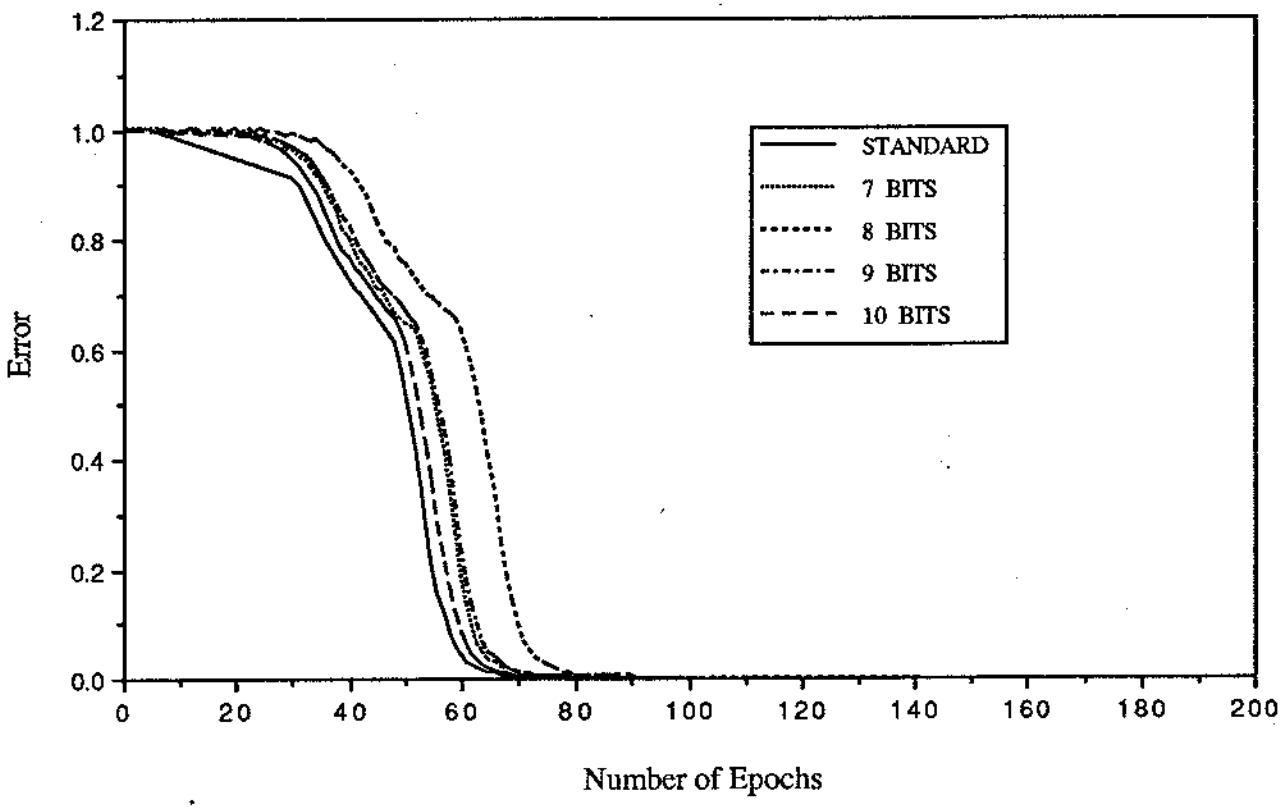
(a) Fixed-point communication.

Figure 4. Fixed and floating-point communication for network A.



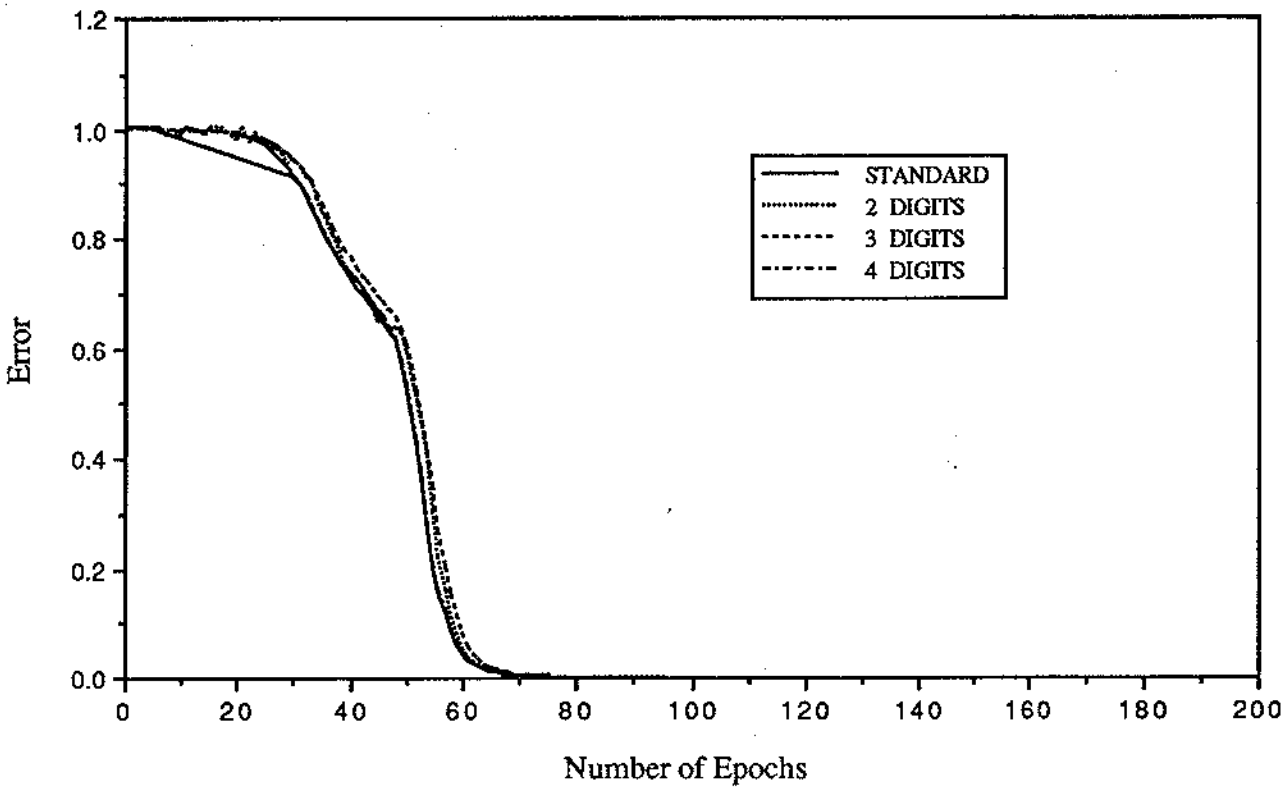
(b) Floating-point communication.

Figure 4. Fixed and floating-point communication for network A (cont.).



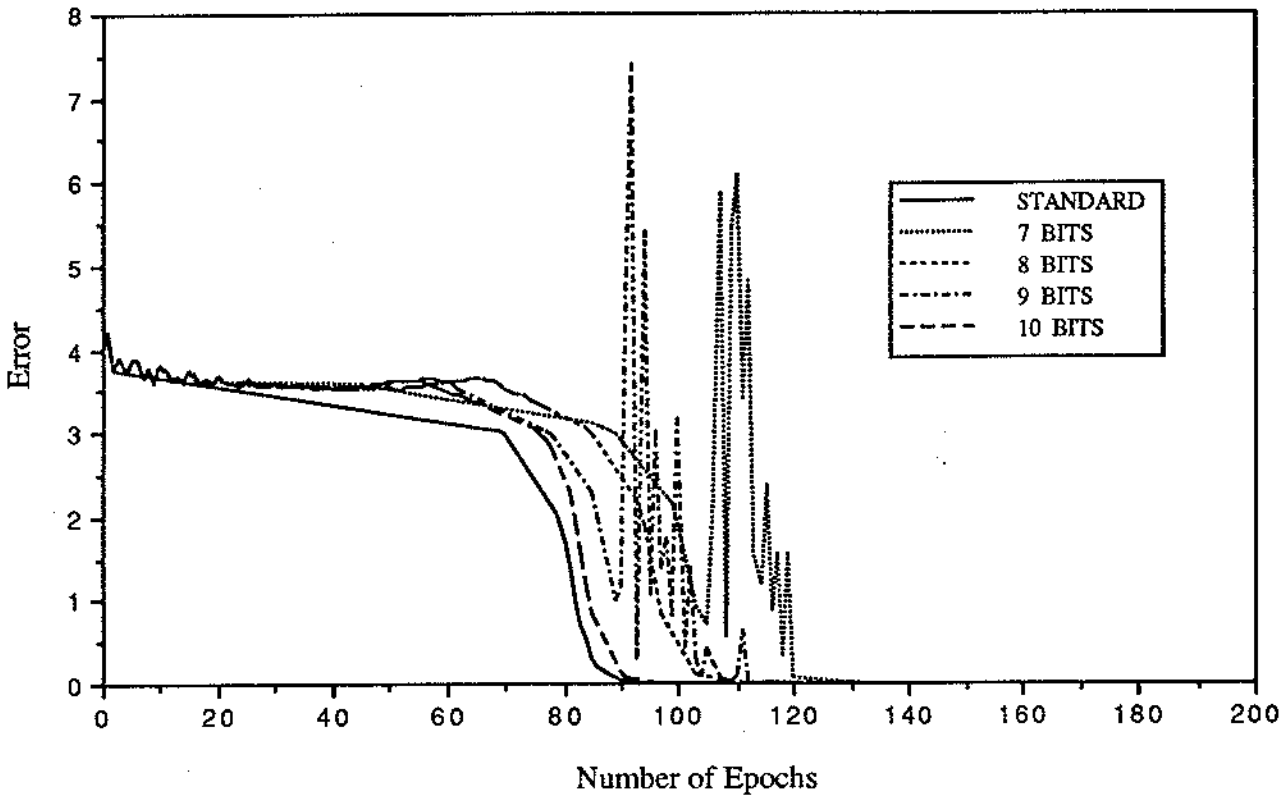
(a) Fixed-point communication.

Figure 5. Fixed and floating-point communication for network B.



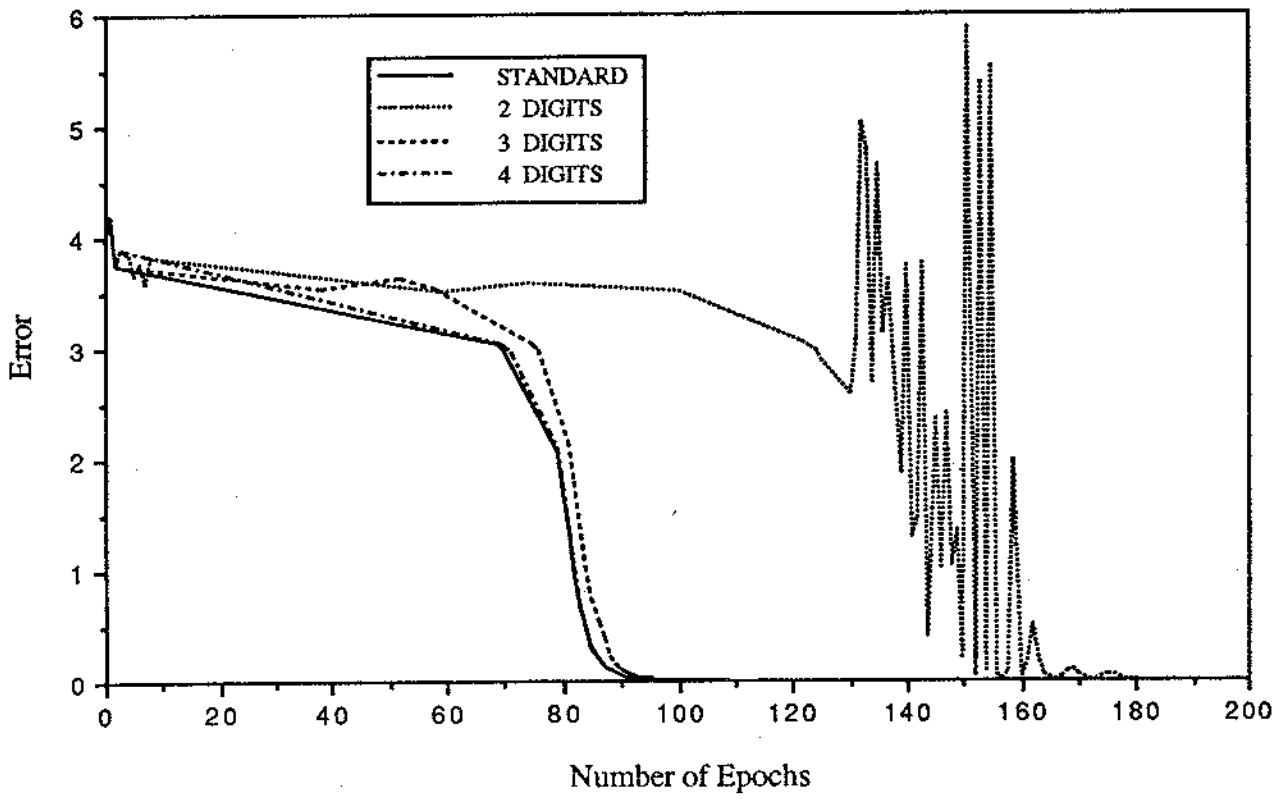
(b) Floating-point communication.

Figure 5. Fixed and floating-point communication for network B (cont.).



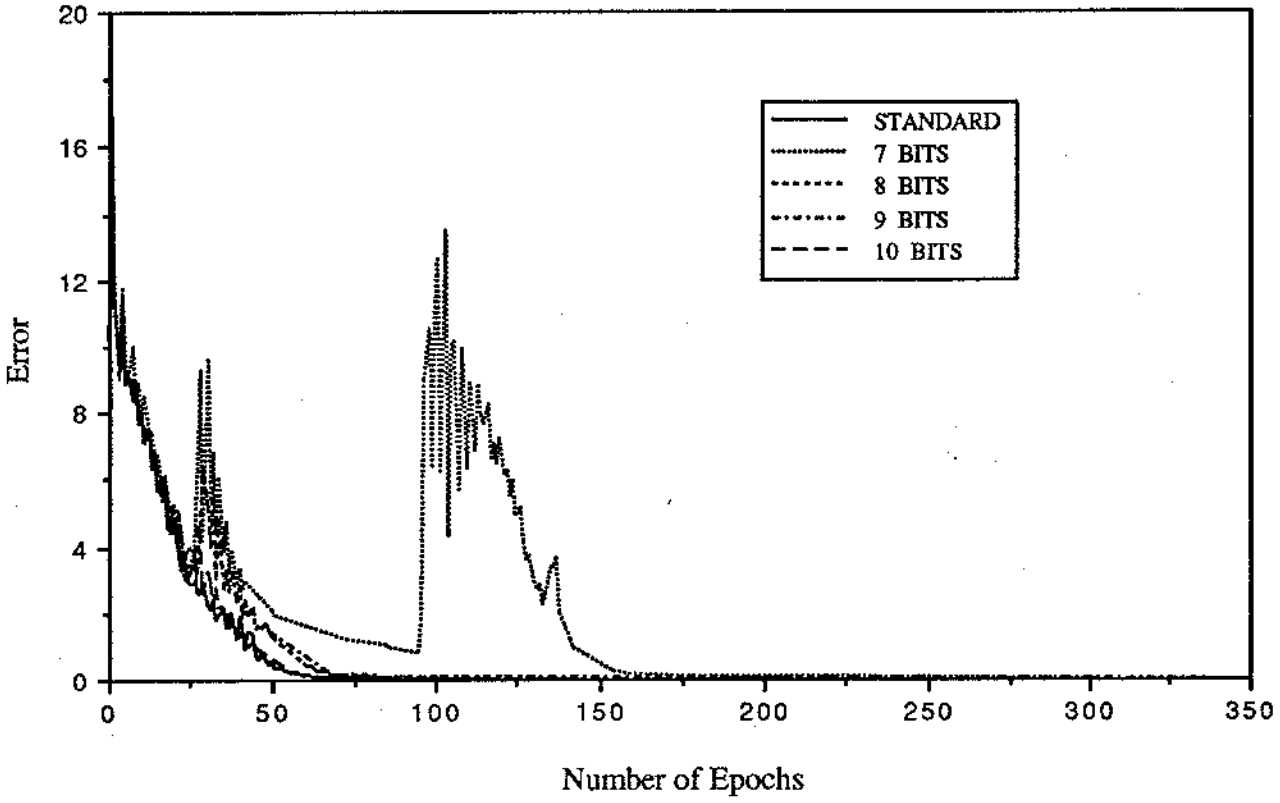
(a) Fixed-point communication.

Figure 6. Fixed and floating-point communication for network C.



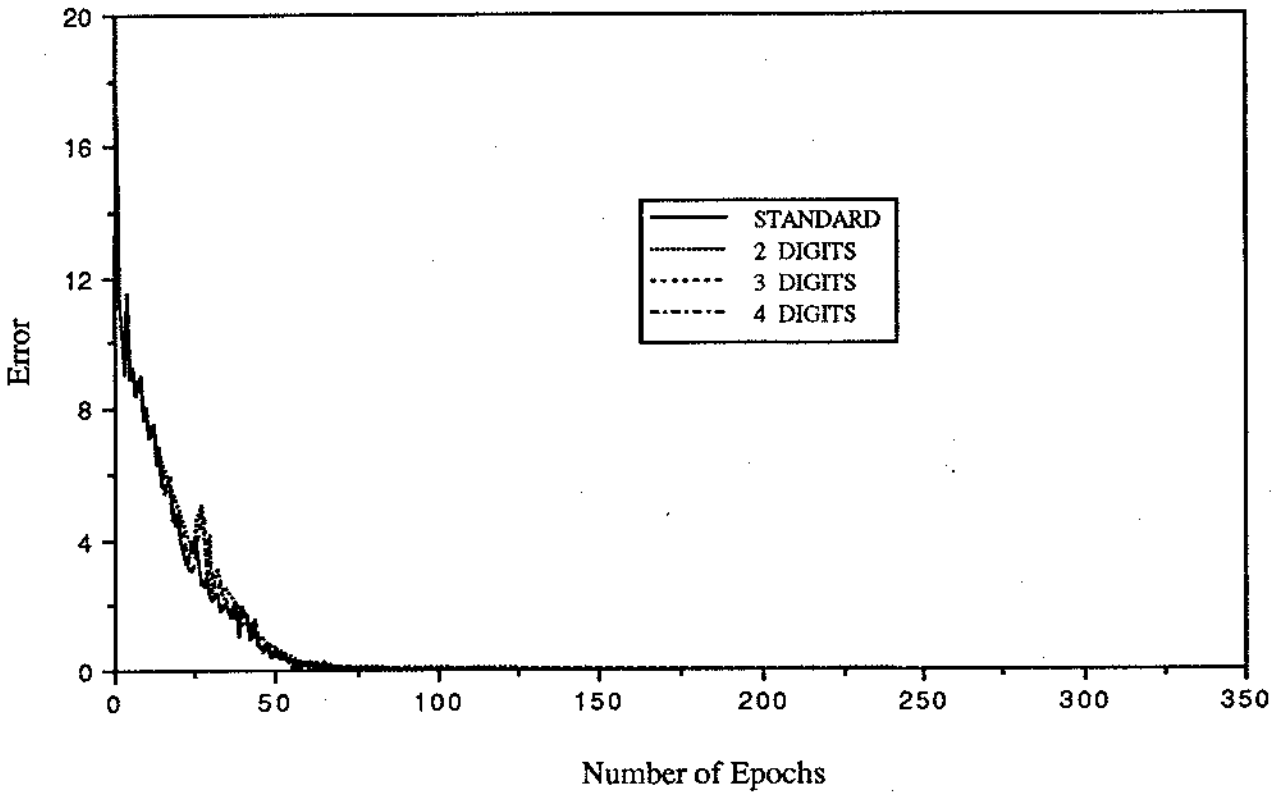
(b) Floating-point communication.

Figure 6. Fixed and floating-point communication for network C (cont.).



(a) Fixed-point communication.

Figure 7. Fixed and floating-point communication for network D.



(b) Floating-point communication.

Figure 7. Fixed and floating-point communication for network D (cont.).

Table 1. The number of epochs required for generalization.

TYPE / NETWORK	FLOATING-POINT COMMUNICATION *			FIXED-POINT COMMUNICATION				STAN-DARD COMM.
	2 DIGITS	3 DIGITS	4 DIGITS	7 BITS	8 BITS	9 BITS	10 BITS	
Network A	130	106	97	207	154	116	104	70
Network B	96	87	82	102	141	96	87	70
Network C	229	109	109	199	130	121	108	109
Network D	242	209	184	435	337	336	210	116

* Number of digits in mantissa.