Finite Precision Error Analysis of Neural Network Hardware Implementations

Jordan L. Holt and Jenq-Neng Hwang, *Member, IEEE*

Abstract— **Through parallel processing, low precision fixed point hardware can be used to build a very high speed neural network computing engine where the low precision results in a drastic reduction in system cost. The reduced silicon area required to implement a single processing unit is taken advantage of by implementing multiple processing units on a single piece of silicon and operating them in parallel. The important question which arises is how much precision is required to implement neural network algorithms on this low precision hardware. A theoretical analysis of error due to finite precision computation was undertaken to determine the necessary precision for successful forward retrieving and back-propagation learning in a multilayer perceptron. This analysis can be easily further extended to provide a general finite precision analysis technique by which most neural network algorithms under any set of hardware constraints may be evaluated.**

Index Terms— **Back-propagation learning, central limit theorem, finite precision computation, general compound operator, jamming and rounding errors, neural network hardware, precision error ratio, random variable, soft and hard convergence.**

I. Introduction

THE high computing speed and low cost desired in the implementation of many neural network algorithms, such as back-propagation learning in a multilayer perceptron (MLP), may be attained through the use of low precision hardware [1]. Finite precision hardware, however, is prone to errors. A method of theoretically deriving and statistically evaluating this error is presented and its use demonstrated in a guide to the precision requirements for the back-propagation algorithm. The paper is devoted to the derivation of the techniques involved as well as the details of the back-propagation example. The intent is to provide a general technique by which most neural network algorithms under any set of hardware constraints may be evaluated [2].

Section II demonstrates the sources of error due to finite precision computation and their statistical properties. A general error model is also derived by which an equation for the error at the output of a general compound operator may be written. Error equations are derived, in Section III, for each of the operations required in the forward retrieving and error back-propagation steps of an MLP. Statistical analysis and simulation results of the resulting distribution of errors for each individual step of an MLP are also included in this section. These error equations are then integrated, in Section IV, to predict the influence of finite precision computation on several stages (early, middle, final stages) of back-propagation learning. Finally, concluding remarks are given in Section V.

Manuscript received February 5, 1991; revised August 21, 1991 and April 27, 1992.

J. L. Holt is with Adaptive Solutions, Inc., Beaverton, OR 97006.

J.-N. Hwang is with the Information Processing Laboratory, Department of Electrical Engineering, FT-10, University of Washington, Seattle, WA 98195.

IEEE Log Number 9205429.

II. Sources of Error in Finite Precision Computation

For a finite precision computation of a nonlinear operation of multiple variables, several sources of error exist. For example, in the computation of $y = \phi(wx)$, the two input variables, w and x, have input errors ϵ_w and ϵ_x, respectively, whose sources are prior finite precision data manipulations. There are errors generated due to the finite precision computation of involved operators. More specifically, the finite precision *multiplication* of the two variables generates one error, ϵ_*. Similarly, the finite precision *nonlinear operator* ϕ generates the other error, ϵ_ϕ. Therefore, the resulting finite precision result $\tilde{y}$ is equal to

$$\begin{aligned}\tilde{y} &\equiv \phi((w+\epsilon_w)(x+\epsilon_x)+\epsilon_*)+\epsilon_\phi \\ &= \phi(wx + w\epsilon_x + x\epsilon_w + \epsilon_w\epsilon_x + \epsilon_*) + \epsilon_\phi \\ &\approx \phi(wx + w\epsilon_x + x\epsilon_w + \epsilon_*) + \epsilon_\phi \\ &\approx \phi(wx) + (w\epsilon_x + x\epsilon_w + \epsilon_*)\phi'(wx) + \epsilon_\phi \qquad (1)\end{aligned}$$

where we assume that the error product $\epsilon_w\epsilon_x$ is negligible, and a first-order Taylor series approximation is used.

The input errors are propagated through the operators. For example, the multiplication of the two variables with finite precision errors propagates error $w\epsilon_x + x\epsilon_w$. This propagated error along with the generated finite precision multiplication error, ϵ_*, further propagates through the nonlinear operator, resulting in the total finite precision error,

$$\epsilon_y = (w\epsilon_x + x\epsilon_w + \epsilon_*)\phi'(wx) + \epsilon_\phi. \qquad (2)$$

The total finite precision error ϵ_y imposed on y will then become the input finite precision error of variable y for future operations.

A. Error Generation and Propagation by Successive Operators

A single input compound operator, $y = \Phi(x)$, which is produced by successive operators $\phi_1, \phi_2, \cdots, \phi_n$, is shown in Fig 1. Error at the input, ϵ_x, and error generated in each operator, ϵ_{ϕ_i}, is propagated through the remaining operators to the output. We can approximate the output error, ϵ_y, in terms of ϵ_x, ϵ_{ϕ_i}, and ϕ_i [3]. From Fig. 1,

$$\tilde{y} = \phi_n(\cdots(\phi_2(\phi_1(x+\epsilon_x)+\epsilon_{\phi_1})+\epsilon_{\phi_2})\cdots)+\epsilon_{\phi_n}. \qquad (3)$$

0018–9340/93$03.00 © 1993 IEEE

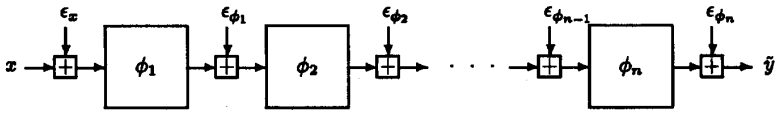

Fig. 1. Successive operators generating and propagating error, where $y = \phi_n(\cdots(\phi_2(\phi_1(x)))\cdots)$.

If y_i is defined as the intermediate result after the first i successive operators,

$$y_i = \phi_i(\phi_{i-1}(\cdots(\phi_1(x))\cdots)) \tag{4}$$

and $y = y_n$, then

$$\begin{aligned}\tilde{y}_i &= \phi_i(\phi_{i-1}(\cdots(\phi_2(\phi_1(x+\epsilon_x)+\epsilon_{\phi_1})+\epsilon_{\phi_2})\cdots) \\ &\quad + \epsilon_{\phi_{i-1}}) + \epsilon_{\phi_i} \\ &= \phi_i(\tilde{y}_{i-1}) + \epsilon_{\phi_i} \\ &= \phi_i(\phi_{i-1}(\tilde{y}_{i-2}) + \epsilon_{\phi_{i-1}}) + \epsilon_{\phi_i} \\ &\approx \phi_i(\phi_{i-1}(\tilde{y}_{i-2})) + \phi_i'(\phi_{i-1}(\tilde{y}_{i-2}))\epsilon_{\phi_{i-1}} + \epsilon_{\phi_i} \\ &\approx \phi_i(\phi_{i-1}(\phi_{i-2}(\tilde{y}_{i-3}))) + \phi_i'\big(\phi_{i-1}(\phi_{i-2}(\tilde{y}_{i-3}))\big) \\ &\quad \cdot \phi_{i-1}'(\phi_{i-2}(\tilde{y}_{i-3}))\epsilon_{\phi_{i-2}} + \phi_i'(\phi_{i-1}(\tilde{y}_{i-2}))\epsilon_{\phi_{i-1}} + \epsilon_{\phi_i} \\ &\approx \quad \vdots \quad \vdots \end{aligned} \tag{5}$$

Carrying out similar expansion for all intermediate values, we can rewrite $\tilde{y}$ to be

$$\begin{aligned}\tilde{y} \equiv \tilde{y}_n &\approx \phi_n(\phi_{n-1}(\cdots(\ \phi_2(\phi_1(x))\ )\cdots)) \\ &\quad + \sum_{i=0}^{n} \epsilon_{\phi_i} \prod_{k=i+1}^{n} \phi_k'(\phi_{k-1}(\cdots(\phi_{i+1}(\tilde{y}_i))\cdots))\end{aligned} \tag{6}$$

where $\prod_{k=n+1}^{n} \phi_k'(\cdot)$ is defined to be 1.

The product shown is just the chain rule for the derivative, $\partial y/\partial \tilde{y}_i$, which can be further approximated by the derivative without error, $\partial y/\partial y_i$. Note that this approximation is equivalent to the approximation already made in the first-order Taylor series.

$$\begin{aligned}\frac{\partial y}{\partial \tilde{y}_i} &= \prod_{k=i+1}^{n} \phi_k'(\phi_{k-1}(\cdots(\phi_{i+1}(\tilde{y}_i))\cdots)) \\ &\approx \prod_{k=i+1}^{n} \phi_k'(\phi_{k-1}(\cdots(\phi_{i+1}(y_i))\cdots)) = \frac{\partial y}{\partial y_i}.\end{aligned} \tag{7}$$

Therefore,

$$\tilde{y} \equiv y + \epsilon_y \approx y + \epsilon_x \frac{\partial y}{\partial x} + \sum_{i=1}^{n} \epsilon_{\phi_i} \frac{\partial y}{\partial y_i}. \tag{8}$$

B. Error Generation and Propagation by General Compound Operators

The effects of finite precision error at the output of a general system of compound operators with multiple input variables can be calculated through an extension of the previous analysis for successive operators of a single variable [3]. This is accomplished by first breaking the computation into a *calculation graph* (see the example given in Fig. 2). The general *calculation graph* is made up of n operators, $\{\phi_i\}$, and m system inputs, $\{x_j\}$. Next, number the operators $\phi_1, \phi_2, \cdots \phi_n$, such that the intermediate generated inputs, $\{y_k\}$, to an operator, ϕ_i, have lower indexes than the operator output, y_i.

By extending (8) to multiple inputs, the total finite precision error, ϵ_y, is given as [3]

$$\epsilon_y \approx \sum_{j=1}^{m} \epsilon_{x_j} \frac{\partial y}{\partial x_j} + \sum_{i=1}^{n} \epsilon_{\phi_i} \frac{\partial y}{\partial y_i}. \tag{9}$$

Using the *calculation graph*, the partial derivatives, $\{\partial y/\partial x_j\}$ and $\{\partial y/\partial y_i\}$, are evaluated and substituted into (9) to give an equation for ϵ_y. Methods include the computation of mean and variance for various functions of random variables, as well as approximations using the central limit theorem.

C. Statistical Evaluation Techniques for Finite Precision Computations

Three common methods used for finite precision computation were analyzed: truncation, conditional jamming, and standard rounding. The truncation operator simply chops the q lowest order bits off of a number and leaves the new lowest order bit, in the 2^rth place, unchanged. The conditional or zero-mean jamming operator[1] chops off the q lowest order bits of a number and forces the new lowest order bit, in the 2^rth place, to be a "1" if any of the q removed bits were a "1"; otherwise, the new lowest order bit retains its value. This operation is equivalent to replacing the 2^rth bit with the logical OR of the 2^rth bit and the q bits which are chopped off. The jamming operator has the advantage of generating error with zero mean, but generates error with a higher variance than that generated by rounding or truncation. The rounding operator also chops off the q lowest order bits of a number which will have its new lowest order bit in the 2^rth place. If the q bit value chopped off is greater than or equal to 2^{r-1}, the resulting value is incremented by 2^r; otherwise, it remains unchanged [1].

The error generated by truncating, jamming, or rounding techniques maybe considered to be a discrete random variable distributed over a range determined by the specific technique being employed. The following assumptions are made:

- Errors are random variables with uniform distributions except as noted.
- Errors are independent of each other and of all inputs and outputs.

For a statistical view of the error, it is desirable to know the mean and variance of the error generated by each of these three techniques. For a discrete random variable, x, the mean

[1] Patent Pending, Adaptive Solutions, Inc., 1990.

is given by

$$\mu = E[x] = \sum_i p_i x_i \tag{10}$$

and the variance is

$$\sigma^2 = E[(x-\mu)^2] = \sum_i p_i (x_i - \mu)^2 \tag{11}$$

where $p_i = Pr\{x = x_i\}$ is usually assumed to be uniformly distributed.

Truncation: Truncation generates error which is uniformly distributed in the range $[-2^r + 2^{r-q},\ 0]$ with each of the 2^q possible error values being of equal probability. Therefore, $p_i = 2^{-q}$ and $x_i = -i \cdot 2^{r-q}$, for $i = 0$ to $2^q - 1$. Mean and variance may then be computed.

$$\mu = -\sum_{i=0}^{2^q-1} 2^{-q} \cdot i \cdot 2^{r-q} = -2^{r-2q} \sum_{i=0}^{2^q-1} i = -\frac{2^r - 2^{r-q}}{2} \tag{12}$$

$$\sigma^2 = \sum_{i=0}^{2^q-1} 2^{-q} \left(-i \cdot 2^{r-q} + \frac{2^r - 2^{r-q}}{2}\right)^2 = 2^{2r} \cdot \frac{1 - 2^{-2q}}{12}. \tag{13}$$

Jamming: The error generated by jamming is not exactly uniformly distributed as the probability of the error being zero is twice the probability of the error holding any of its other possible values. The range of error is $[-2^r + 2^{r-q},\ 2^r - 2^{r-q}]$ with

$$p_i = \begin{cases} 2^{-q-1}, & i \neq 0 \\ 2^{-q}, & i = 0 \end{cases}$$

and $x_i = i \cdot 2^{r-q}$, for $i = -2^q + 1$ to $2^q - 1$. This results in $2^{q+1} - 1$ possible error values. The mean and variance of jamming error are then

$$\mu = \sum_{i=-2^q+1}^{2^q-1} p_i \cdot i \cdot 2^{r-q} = 2^{-q-1} \cdot 2^{r-q} \sum_{i=-2^q+1}^{-1} i + 2^{-q-1} \cdot 2^{r-q} \sum_{i=1}^{2^q-1} i = 0 \tag{14}$$

$$\sigma^2 = \sum_{i=-2^q+1}^{2^q-1} p_i \cdot \left(i \cdot 2^{r-q}\right)^2 = 2^{2r} \cdot \frac{2 - 3 \cdot 2^{-q} + 2^{-2q}}{6}. \tag{15}$$

Rounding: Rounding generates error which is uniformly distributed in the range $[-2^{r-1},\ 2^{r-1} - 2^{r-q}]$ with each of the 2^q possible error values being of equal probability. Therefore, $p_i = 2^{-q}$ and $x_i = i \cdot 2^{r-q}$, for $i = -2^{q-1}$ to $2^{q-1} - 1$. The mean and variance are computed.

$$\mu = \sum_{i=-2^{q-1}}^{2^{q-1}-1} 2^{-q} \cdot i \cdot 2^{r-q} = 2^{r-2q} \sum_{i=-2^{q-1}}^{2^{q-1}-1} i = -2^{r-q-1} \tag{16}$$

$$\sigma^2 = \sum_{i=-2^{q-1}}^{2^{q-1}-1} 2^{-q} \left(i \cdot 2^{r-q} + 2^{r-q-1}\right)^2 = 2^{2r} \cdot \frac{1 - 2^{-2q}}{12}. \tag{17}$$

Nonlinear Functions of Discrete Random Variables: For a nonlinear function of a discrete random variable, x, the mean and variance are given by

$$\mu = E[f(x)] = \sum_i p_i \cdot f(x_i) \tag{18}$$

$$\sigma^2 = E[(f(x) - \mu)^2] = \sum_i p_i \cdot (f(x_i) - \mu)^2. \tag{19}$$

D. Statistical Properties of Independent Random Variables

For two independent random variables, x and y, with means μ_x, μ_y and variances σ_x^2, σ_y^2, and a constant a, the following properties of mean and variance can be shown [6].

1) Multiplication by a constant.

$$\mu_{ax} = a\mu_x, \qquad \sigma_{ax}^2 = a^2 \sigma_x^2. \tag{20}$$

2) Sum of two independent random variables.

$$\mu_{x+y} = \mu_x + \mu_y, \qquad \sigma_{x+y}^2 = \sigma_x^2 + \sigma_y^2. \tag{21}$$

3) Product of two independent random variables.

$$\mu_{xy} = \mu_x \mu_y, \qquad \sigma_{xy}^2 = \sigma_x^2 \sigma_y^2 + \sigma_x^2 \mu_y^2 + \sigma_y^2 \mu_x^2. \tag{22}$$

E. Statistical Properties of Sums of Independent Random Variables

Central Limit Theorem: The central limit theorem [6] states that if $\{x_i\}$ are N independent random variables, then the density of their sum, properly normalized, tends to a normal curve as $N \to \infty$. For discrete random variables, the probabilities tend to the samples of a normal curve. Normalization can be achieved in a couple of different ways. If $\{x_i\}$ are discrete random variables with mean μ and variance σ^2, then the sum $\overline{x}$;

$$\overline{x} = \sum_{i=1}^{N} x_i \quad \text{results in} \quad \mu_{\overline{x}} = N\mu, \text{ and } \sigma_{\overline{x}}^2 = N\sigma^2. \tag{23}$$

Invoking the central limit theorem, the probability that the sum of the random variables, $\overline{x}$, is equal to the discrete value x_k which approaches the sample of a normal curve when N is large.

$$p_k \equiv \Pr\{\overline{x} = \overline{x}_k\} \simeq \frac{1}{\sigma_{\overline{x}}\sqrt{2\pi}} e^{-(\overline{x}_k - \mu_{\overline{x}})^2 / 2\sigma_{\overline{x}}^2}. \tag{24}$$

Sum of Products of Independent Random Variables: The central limit theorem can be extended to cover the case where the random variable being summed is a product of random variables. Say that for independent random variables $\{x_i\}$ and $\{y_i\}$,

$$\overline{xy} = \sum_{i=1}^{N} x_i y_i \tag{25}$$

then the probability density of the random variable $\overline{xy}$ approaches a normal curve for large N with mean and variance equal to

$$\mu_{\overline{xy}} = N\mu_{xy}, \quad \text{and} \quad \sigma_{\overline{xy}}^2 = N\sigma_{xy}^2. \tag{26}$$

Expected Squared Error: The expected squared error can be written in terms of the mean and variance of the error. Consider a set of errors which are independent random variables, $\{\epsilon_i\}$, with mean μ and variance σ^2. Then, the expected value of the average sum of the squares of $\{\epsilon_i\}$ can be written

$$E\left[\frac{1}{N}\cdot\sum_{i=1}^{N}\epsilon_i^2\right]=\frac{1}{N}\cdot N\cdot E[\epsilon_i^2]=E[\epsilon_i^2]. \quad (27)$$

Noting that $\sigma^2 = E[\epsilon_i^2] - E^2[\epsilon_i]$, the expected value of the average sum squared errors is equal to

$$E[\epsilon_i^2]=\sigma^2+\mu^2. \quad (28)$$

F. Summary of Finite Precision Error Analysis Technique

1) Through the *calculation graph* technique of Section II-B, the finite precision error at the output of a general compound operator can be written:

$$\epsilon_y \approx \sum_{j=1}^{m}\epsilon_{x_j}\frac{\partial y}{\partial x_j}+\sum_{i=1}^{n}\epsilon_{\phi_i}\frac{\partial y}{\partial y_i}. \quad (29)$$

2) The statistical techniques in Sections II-C to II-E are used to determine the expected squared value of this error.

$$\begin{aligned}E[\epsilon_y^2] &= E^2[\epsilon_y]+E[(\epsilon_y-E[\epsilon_y])^2]\\ &= \mu_{\epsilon_y}^2+\sigma_{\epsilon_y}^2.\end{aligned} \quad (30)$$

3) Successive stages of general compound operators (say y is the input to another compound operator) are handled by using μ_{ϵ_y} and σ_{ϵ_y} to describe the finite precision error distribution of y as an input variable.

III. Application to Neural Network Retrieving and Learning

It has been shown that operations in both the retrieving and the learning phases of most of the neural network models can be formulated as a linear *affine transformation* interleaved with simple linear or nonlinear scalar operations. In terms of the hardware implementations, all these formulations call for MAC (multiply and accumulation) processor hardware [1], [4], [5]. The performance of parallel MAC processor hardware can be greatly improved and the cost significantly reduced if the implementation is done in low precision hardware. Without loss of generality, we have chosen to specifically discuss the multilayer perceptron neural network model and back-propagation learning [7], [8] as these algorithms are the most commonly used.

Note that for back-propagation learning, this technique measures the deviation from gradient descent search which is itself an errorful process. The evaluation of the gradient descent approximation is widely studied [8] and beyond the scope of this paper.

A. Forward Retrieving and Back Propagation of an MLP

Given a trained (L-layer, fixed-weight) MLP, the retrieving phase receives the input test pattern, $\{x_{0,i}\}$, and propagates forward through the network to compute the activation values of the output layer, $\{x_{L,j}\}$, which will be used as the indicator for classification or regression purposes. On the other hand, in the commonly used learning phase of an MLP, the input training pattern, $\{x_{0,i}\}$, is first propagated forward through the network and the activation values, $\{x_{l,j}\}$, are computed according to the same forward operations used in the retrieving phase. Then the output activation values, $\{x_{L,j}\}$, are compared with the target values $\{t_j\}$; and the value of the output delta, $\{\delta_{L,j}\}$, for each neuron in the output layer is derived. These error signals are propagated backward to allow the recursive computation of the hidden delta's, $\{\delta_{l,j}\}$, as well as the update values of the weights, $\{\Delta w_{l,i,j}\}$, at each layer. While many methods have been proposed to accelerate learning by estimating the local curvature of the training error surface using second order derivative information, the discussion of these methods are beyond the scope of this paper, and can be referred to [9].

The operations in the forward retrieving of an L-layer perceptron can be formulated as a *forward affine transformation* interleaved with a nonlinear scalar activation function:

$$x_{l+1,j}=f\Big(\sum_i w_{l+1,i,j}\cdot x_{l,i}\Big), \qquad \forall\, j, \quad \forall\, l \quad (31)$$

where $x_{l,i}$ denotes the activation value of the ith neuron at the lth layer, $w_{l+1,i,j}$ denotes the synaptic weight interconnecting the ith neuron at the lth layer and the jth neuron at the $(l+1)$th layer. The nonlinear activation function, $f(\cdot)$, is usually taken to be sigmoidal ($f(y) = 1/(1+e^{-y})$). It is assumed that one of the synaptic weights of each neuron represents that neuron's bias or threshold (say: $x_{l,1} \equiv 1.0$ and $w_{l+1,1,j} \equiv \theta_{l+1,j}$).

The back-propagation training of an MLP follows the iterative gradient descent approach with the following update at each presentation of a training data pair [7], [8]:

$$\Delta w_{l,i,j}=\eta\cdot\delta_{l,j}\cdot x_{l-1,i} \quad (32)$$

where the computation of the back-propagated error, $\delta_{l,j}$, can again be formulated as a *backward affine transformation* interleaved with the post-multiplication of the derivatives of the nonlinear activation function:

$$\delta_{l,j}=f'_{l,j}\cdot\sum_i \delta_{l+1,i}\cdot w_{l+1,j,i} \quad (33)$$

with initial output-layer propagated error being

$$\delta_{L,j}=f'_{L,j}\cdot(t_j-x_{L,j}). \quad (34)$$

B. Finite Precision Analysis of Forward Retrieving

Explicitly following the procedure discussed in Section II-B, the calculation graph of the forward retrieving operation, with simplified notation [see (31)], in an MLP is shown in Fig. 2. Again, it is assumed that the bias or threshold term is taken care of by a constant at one of the inputs (say $x_1 \equiv 1.0$,

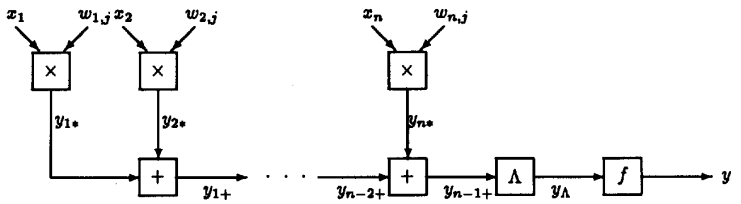

Fig. 2. Calculation graph for the forward retrieving, $y = x_{l+1,j}$, of an MLP, where Λ denotes a truncation, jamming, or rounding operator.

then $w_{1,j} \equiv \theta_j$).

$$y = f(\sum_i w_{i,j} \cdot x_i). \tag{35}$$

To carry out the analytical formula as given in (9) for the forward retrieving of an MLP, several partial derivatives need to be computed from (35):

$$\frac{\partial y}{\partial x_i} = f'_j\, w_{i,j}, \text{ and } \frac{\partial y}{\partial w_{i,j}} = f'_j\, x_i \tag{36}$$

where $f'_j = y(1-y)$ due to the nice property of the sigmoid function.

$$\frac{\partial y}{\partial y_{i*}} = \frac{\partial f(\sum_{i=1}^{n} y_{i*})}{\partial y_{i*}} = f'_j \tag{37}$$

$$\frac{\partial y}{\partial y_{i+}} = \frac{\partial f(y_{i+} + \sum_{k=i+1}^{n} y_{k*})}{\partial y_{i+}} = f'_j \tag{38}$$

$$\frac{\partial y}{\partial y_\Lambda} = \frac{\partial f(y_\Lambda)}{\partial y_\Lambda} = f'_j. \tag{39}$$

By substituting the values for the partial derivatives in (36) to (39) and the generated and propagated errors for variables and operators into (9),

$$\epsilon_y \equiv \epsilon_{x_{l+1,j}} = f'_j \sum_{i=1}^{n} w_{i,j}\epsilon_{x_i} + f'_j \sum_{i=1}^{n} x_i \epsilon_{w_{i,j}} + f'_j \sum_{i=1}^{n} \epsilon_{y_{i*}} + f'_j \sum_{i=1}^{n-1} \epsilon_{y_{i+}} + f'_j \epsilon_{y_\Lambda} + \epsilon_{f_j}. \tag{40}$$

C. Finite Precision Analysis of Output Delta

From (34), the calculation graph for the computation of back-propagated error in an output neuron, with simplified notation, is shown in Fig. 3.

$$y = f'_j \cdot (t_j - x_j) = x_j(1-x_j) \cdot (t_j - x_j). \tag{41}$$

Again, to carry out the analytical formula as given in (9) for the output delta computation of an MLP, the partial derivatives of (41) are evaluated.

$$\frac{\partial y}{\partial x_j} = -f'_j + (t_j - x_j)\frac{\partial f'_j}{\partial x_j} = -f'_j + (t_j - x_j)(1-2x_j) \tag{42}$$

$$\frac{\partial y}{\partial t_j} = f'_j \tag{43}$$

$$\frac{\partial y}{\partial y_{f'}} = t_j - x_j \tag{44}$$

$$\frac{\partial y}{\partial y_{j-}} = f'_j. \tag{45}$$

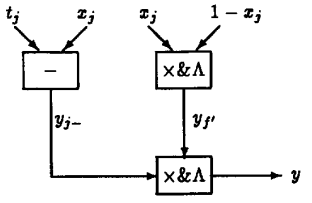

Fig. 3. Calculation graph for $y = \delta_{L,j}$ in an output neuron.

Substituting for the partial derivatives and individual error terms, the overall finite precision error for the output delta computation is

$$\epsilon_y \equiv \epsilon_{\delta_{L,j}} = \left(-f'_j + (t_j - x_j)(1-2x_j)\right)\epsilon_{x_j} + f'_j\epsilon_{t_j} + (t_j - x_j)\epsilon_{f'_j} + f'_j\epsilon_{y_{j-}} + \epsilon_{*\Lambda}. \tag{46}$$

D. Finite Precision Analysis of Hidden Delta

From (33), the calculation graph for the computation of back-propagated error in a hidden layer neuron, with simplified notation, is shown in Fig. 4.

$$y = f'_j \cdot \sum_i^n \delta_i \cdot w_{j,i} = x_j(1-x_j) \cdot \sum_i^n \delta_i \cdot w_{j,i}. \tag{47}$$

Following similar partial derivative evaluations using (49), we can again compute the finite precision error for the hidden delta [see (9)]:

$$\epsilon_y \equiv \epsilon_{\delta_{l,j}} = f'_j \sum_{k=1}^{n} w_{j,k}\epsilon_{\delta_k} + f'_j \sum_{k=1}^{n} \delta_k \epsilon_{w_{j,k}} + (1-2x_j)\epsilon_{x_j} \sum_{k=1}^{n} \delta_k w_{j,k} + f'_j \sum_{k=1}^{n} \epsilon_{y_{k*}} + \epsilon_{y_{f'}} \sum_{k=1}^{n} \delta_k w_{j,k} + f'_j \sum_{k=1}^{n-1} \epsilon_{y_{k+}} + f'_j\epsilon_\Lambda + \epsilon_{\times\Lambda}. \tag{48}$$

E. Finite Precision Analysis of Weight Update

From (32), the calculation graph for the computation of the weight update (without the momentum term), with simplified notation, is shown in Fig. 5.

$$y = \eta \cdot \delta_j \cdot x_i \tag{49}$$

Following similar partial derivative evaluations using (47), we can again compute the finite precision error for weight

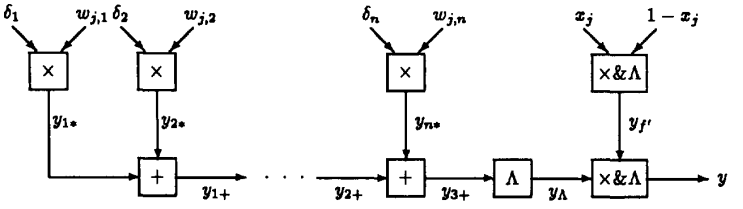

Fig. 4. Calculation graph for $y = \delta_{l,j}$ in a hidden neuron.

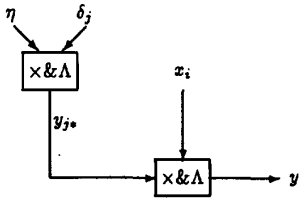

Fig. 5. Calculation graph for $y = \Delta w_{l,i,j}$.

update [see (9)]:

$$\epsilon_y \equiv \epsilon_{\Delta w_{l,i,j}} = \delta_j x_i \epsilon_\eta + \eta x_i \epsilon_{\delta_j} + \eta \delta_j \epsilon_{x_i} + x_i \epsilon_{y_{j*}} + \epsilon_{*\Lambda}. \quad (50)$$

F. Statistical Evaluation of the Finite Precision Errors

Given the analytical expressions of all the finite precision errors associated with the forward retrieving and back-propagation of an MLP, a statistical evaluation of these errors is undertaken. The second and third steps in the *finite precision error analysis technique* of Section II-F are now used to statistically evaluate the error equations derived above. These steps are based on the techniques shown in Sections II-C, II-D, and II-E.

Two precisions were varied in this analysis, the activation precision and the weight precision where each refers to a number of individual components (see Table I):

- *Activation precision* ($\equiv \boldsymbol{A}$ bits) refers to the precision of the inputs, targets, outputs, and lookup table input and output (activation).
- *Weight precision* ($\equiv \boldsymbol{W}$ bits) refers to the precision of the weights, biases, weight updates, output δ, and hidden δ.

For the following analysis, the range of each variable was fixed. This fixed range was chosen based on typical ranges suggested in [10]. The precision for both the activation and the weights is independently varied between 3 and 16 bits for the forward retrieving analysis. The precision for the activation is varied between 4 and 24 bits for the back-propagation analysis while the precision for the weights is independently varied between 8 and 24 bits.

Finite Precision Error in Forward Retrieving: The expected forward retrieving error can be calculated for both a single layer of neurons and for multiple layers of neurons by propagating upward the finite precision errors of the lower layers. First, a simplification may be made to (40). The multiply and accumulate steps can be computed without generating any error if enough bits (e.g., $\boldsymbol{W} + \boldsymbol{A}$ bits) are used for each of the intermediate variables, y_{i*} and y_{i+}. In this case, $\epsilon_{y_{i*}} = \epsilon_{y_{i+}} = 0$. This is practical since the expense of accumulator precision is very small.

The Λ operator now reduces the final $(\boldsymbol{W} + \boldsymbol{A})$-bit sum to an $\boldsymbol{A}$-bit (one sign bit, 3 bits to the left, and $\boldsymbol{A} - 4$ bits to the right of the decimal) value which is used as the input for the sigmoid lookup table. Equation (40) may now be rewritten as

$$\epsilon_y = f_j' \sum_{i=1}^{n} w_{ij} \epsilon_{x_i} + f_j' \sum_{i=1}^{n} x_i \epsilon_{w_{ij}} + f_j' \epsilon_\Lambda + \epsilon_{f_j}. \quad (51)$$

Before computing the distribution of the sums, it is necessary to know the distributions of the random variables w_{ij}, $\epsilon_{w_{ij}}$, x_i, ϵ_{x_i}, $\epsilon_{f_j'}$, ϵ_Λ, and ϵ_{f_j}. Table II shows each contributing component to (51) based on $\boldsymbol{W}$ bits of weight precision and $\boldsymbol{A}$ bits of activation precision. The evaluation starts with $\boldsymbol{W}$-bit weights which are uniformly distributed across the entire range, $[-8, 8)$, and weight error comes from truncating 24-bit weights to $\boldsymbol{W}$ bits. For an input neuron, x_i is an $\boldsymbol{A}$-bit value uniformly distributed over $[0.0, 1.0)$ which has been truncated from a 24-bit value. The distribution of f_j' is approximated as the sigmoid derivative function of a normally distributed random variable whose mean and variance were experimentally estimated. ϵ_Λ is the error generated when the accumulated $(\boldsymbol{W} + \boldsymbol{A})$-bit value is jammed to become an $\boldsymbol{A}$-bit value, and ϵ_{f_j} is the error generated in the lookup table which occurs when the actual table is rounded by $\boldsymbol{A}$ bits at the $2^{-\boldsymbol{A}}$th place.

The following theoretical finite precision error predictions are based on a network topology, which is a single hidden layer MLP with 100 inputs, 100 hidden units, and 100 outputs. It was found that the error varies very slowly with the number of units assumed in each layer, so the network was chosen to be big enough that the statistical analysis has a large number of samples.

For 3 to 16 bits of activation precision and 3 to 16 bits of weight precision, the expected square forward retrieving error, $E[\epsilon_y^2]$, is calculated for neurons in the first hidden layer (Fig. 6) and for neurons in the second hidden layer or output layer (Fig. 7). An approximation of the number of significant bits at the output of the computation can be found by assuming that the output error has zero mean and a variance like that of rounding. Then, for large q, (16) gives

$$E[\epsilon_y^2] \approx \frac{2^{2r}}{12} \quad (52)$$

or,

$$r \approx \frac{1}{2} \cdot \log_2(E[\epsilon_y^2]) + \frac{1}{2} \cdot \log_2(12) \quad (53)$$

TABLE I
LIMITED PRECISION ENVIRONMENTS TO BE ANALYZED

	Limited Precision Fixed Point		Floating Point
	Number of Bits	Range	Number of Bits
Inputs, outputs, & targets	**A**	[0.0,1.0)	32
Sigmoid function lookup (input)	**A** (Activation)	[8.0, 8.0)	32
Sigmoid function lookup (output)	**A** (Activation)	[0.0, 1.0)	32
Weights, biases, & wt. updates	**W** (Weight)	[−8.0, 8.0)	32
Output δ	**W** (Weight)	[−0.5, 0.5)	32
Hidden δ	**W** (Weight)	[−8.0, 8.0)	32

TABLE II
VARIABLES IN THE FORWARD RETRIEVING CALCULATION WITH $\boldsymbol{W}$ BIT WEIGHTS, BIASES, AND WEIGHT UPDATES AND $\boldsymbol{W}$ BIT INPUTS, OUTPUTS, AND TARGETS

Random Variables

R.V.	Type	Number of Bits	Bits Above Bin. Pt.
w_{ij}	Uniform	$\boldsymbol{W}$	3
x_i	Uniform	$\boldsymbol{A}$	0

Error Random Variables

R.V.	Type	Bits Removed	Highest Bit Removed
$\epsilon_{w_{ij}}$	Rounding	$q = 24 - \boldsymbol{W}$	$r = 4 - \boldsymbol{W}$
ϵ_{x_i}	Truncation	$q = 24 - \boldsymbol{A}$	$r = -\boldsymbol{A}$
ϵ_Λ	Jamming	$q = \boldsymbol{W}$	$r = 4 - \boldsymbol{A}$
ϵ_f	Rounding	$q = \boldsymbol{A}$	$r = -\boldsymbol{A}$

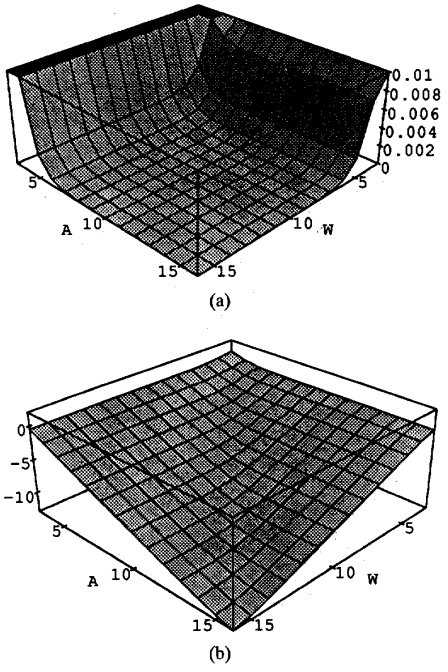

Fig. 6. Expected square forward retrieving error introduced to neurons of the first hidden layer. (a) $E[\epsilon_y^2]$, (b)$(1/2) \cdot \log_2(E[\epsilon_y^2]) + (1/2) \cdot \log_2(12)$.

where r approximates the placement of the highest output bit with high error.

To use the least total precision, the log plot shows that the choice of one precision will dictate the choice of the other

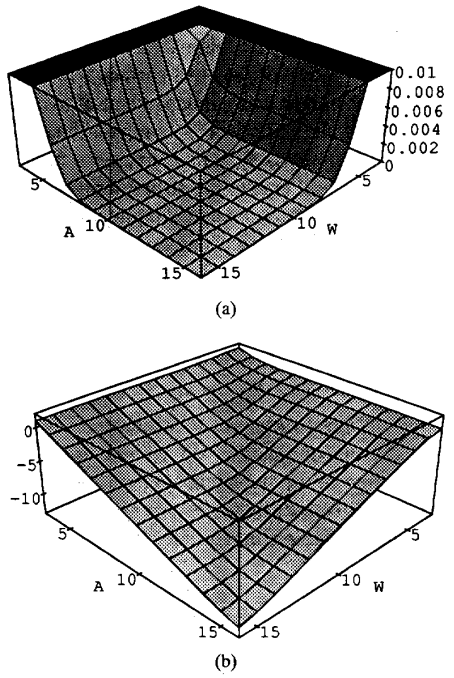

Fig. 7. Expected square forward retrieving error introduced to neurons of the second hidden layer oroutput layer. (a) $E[\epsilon_y^2]$,(b) $\frac{1}{2} \cdot \log_2(E[\epsilon_y^2]) + (1/2) \cdot \log_2(12)$.

precision along the line which joins the two error planes. For forward retrieving, this line is approximately where activation precision is equal to weight precision. For example, the choice of 8 bit activation precision would result in a best choice of weight precision of 8 bits.

The log plot can also be used to choose the necessary precision for a desired significant error at the output. For example, if output error can be tolerated at the 2^{-3} place ($|\epsilon_y| \leq 0.125$) then Fig. 7(b) shows that 8 bits of weight and activation precision are necessary to give a log error below -3.

IV. FINITE PRECISION ANALYSIS FOR ITERATIVE LEARNING

As discussed in Section III-A, back-propagation learning involves all four consecutive steps of computation: forward retrieving, output delta computation, hidden delta computation, and weight updating. Therefore, for each weight updating iteration at the presentation of any training pattern, the finite precision error $\epsilon_{\Delta w}$ introduced to $\{\Delta w_{l,i,j}\}$ given in (50)

is in fact a propagated result of the error generated from the previous three steps. Therefore, the final mathematical expression of finite precision error for a *single weight updating iteration* can be formulated in a straightforward manner based on the existing derivations given in (40), (46), (48), and (50). The statistical evaluation value for the average sum of the squares of weight updating error $\epsilon_{\Delta w}$ due to the finite precision computation of a single learning iteration can thus be computed.

The back-propagation learning discussed above is simply a nonlinear optimization problem based on simple gradient descent search, with an elegant way of computing the gradients using the chain rule on the layers of the network. This gradient descent search updates the weights based only on the first derivative approximation of the error surface with the updating of each individual weight being independent of the others [11]. Therefore, even if the approach is computationally efficient, it can behave very unwisely and converge very slowly for complex error surfaces. Due to the strong influence introduced in the gradient descent search approximation, the real effect to the learning convergence and accuracy due to finite precision computation will be difficult to measure. Therefore, the statistically evaluated average sum of the squares of $\epsilon_{\Delta w}$, by itself, does not determine a network's propensity to learn.

A. Ratio of Finite Precision Errors

A more meaningful measure, ρ, which indicates the effect of finite precision computation on weight updating, can be defined as the ratio of the statistical average sum of the squares of finite precision weight updating error $\epsilon_{\Delta w}$ and that of full precision weight updating magnitude Δw_{ij}:

$$\rho \equiv \frac{E\left[\epsilon_{\Delta w}^2\right]}{E\left[\Delta w_{ij}^2\right]}. \tag{54}$$

This ratio serves as a useful indicator of the additional impact of finite precision computation on the gradient descent approximation of back-propagation learning. The finite precision ratio depends not only on the number of bits assigned to the finite precision computation, but on the current stage of learning progress, which can be specified by the distribution of the difference between the desired and actual output $\{t_j - x_{L,j}\}$. More specifically,

$$\beta = E\left[(t_j - x_{L,j})^2\right] \tag{55}$$

where we assume that $\epsilon_{x_{L,j}} = \epsilon_{t_j} = 0$, since the ability to learn should depend on the ability to learn these finite precision values.

For 4 to 24 bits of activation precision and 4 to 24 bits of weight precision, the finite precision ratio is evaluated for neurons in the output layer (Fig. 8) and for neurons in the hidden layer (Fig. 9) at several different states of learning. Four different values of β are used, which represent four different states of learning as shown in Table III.

Fig. 8 shows the statistical evaluation values of the finite precision ratio for the weights connecting the neurons between the hidden and the output layers in a 2-layer MLP. Fig. 9 shows the statistical evaluation values of the finite precision ratio for the weights connecting the neurons between the hidden and the input layers for these four different values of β.

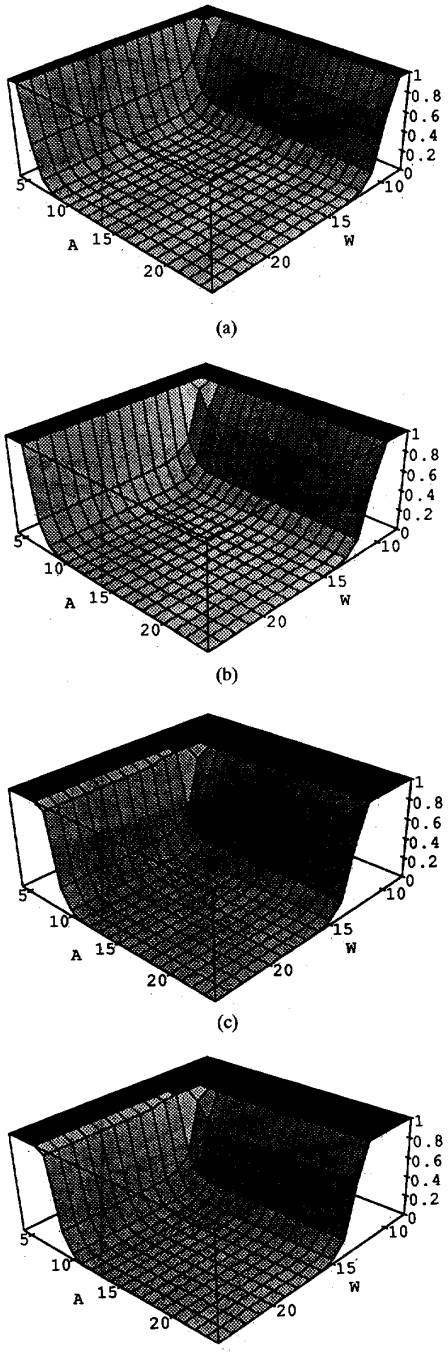

Fig. 8. The finite precision ratio for the top-layer weights in a 2-layer MLP. (a) β = 0.0833, (b) β = 0.0208, (c) β = 0.00521, (d) β = 0.00130.

Assuming a well posed problem, it is expected that learning will take place for $\rho << 1.0$ and learning will fail for $\rho \geq 1.0$. The weight and activation precisions at which the finite precision ratio curves dive quickly below 1.0 indicates

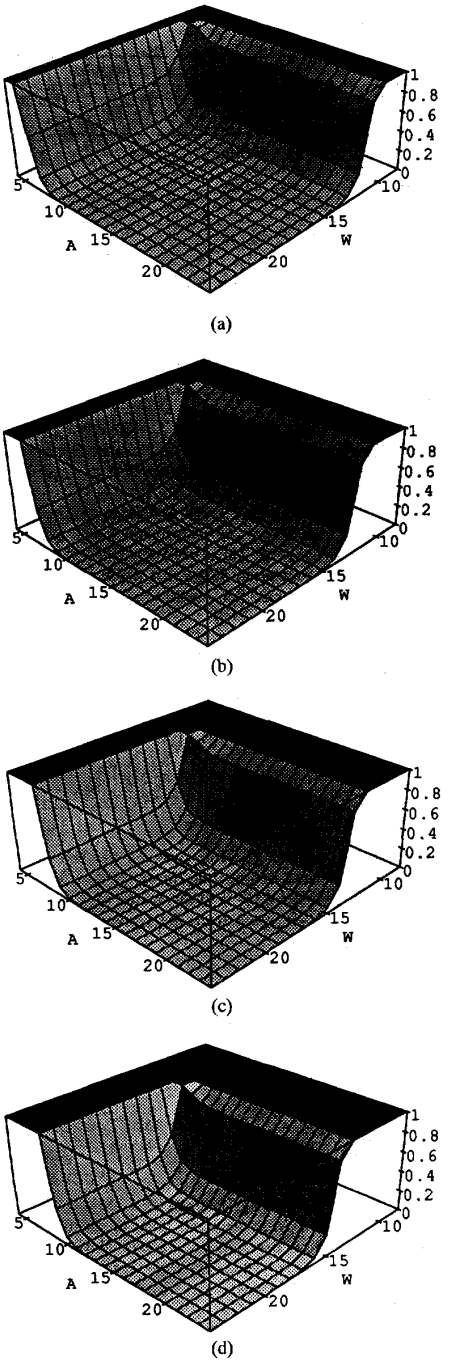

Fig. 9. The finite precision ratio for the hidden-layer weights in a 2-layer MLP. (a) β = 0.0833, (b) β = 0.0208, (c) β = 0.00521, (d) β = 0.00130.

that learning convergence and accuracy will come close to matching that of floating point at precisons above this dive point.

- 13–15 bits of weight precision and 8–9 bits of activation precision for *soft convergence learning.*
- 15–16 bits of weight precision and 9–10 bits of activation precision for *hard convergence learning.*

TABLE III
LEARNING STAGES TO BE ANALYZED

Learning Stage	$\lvert t_j - x_{L,j}\rvert \leq$	β
Early Stage	2^{-1}	0.0833
Middle Stage	2^{-2}	0.0208
Soft Convergence Stage	2^{-3}	0.00521
Hard Convergence Stage	2^{-4}	0.00130

Smaller values of β (tighter learning) require higher precision as the network has to make very small adjustments to its weights. An important point is that the precision requirements for the weights are dominated by the requirements for the weight updates. In learning, it is possible to use weights with the same precision as the activation values as long as a higher precision weight update is accumulated.

B. Simulation Results for Iterative Learning

To verify the theoretical evaluation of back-propagation learning with finite precision computation, a simple regression problem is designed, which maps two-dimensional inputs, $\{x_1, x_2\}$, into one-dimensional outputs, $\{y\}$:

$$y = \frac{1}{2}(x_1 + x_2)^2 \cdot e^{-x_1^2 - x_2^2 + 1}. \tag{56}$$

An MLP containing 2 input neurons, 8 hidden neurons, and 1 output neuron, is adopted. The training data is constructed from 256 pairs of randomly selected data. Finite precision learning simulations were performed with $\boldsymbol{A} = 8$ bits of activation precision and $\boldsymbol{W} = 8$ to 16 bits of weight and weight update precision. Fig. 10(a) shows the average (of 256 training data) squared difference between the desired and actual outputs of the 2-D regression problem after the network converges (a hard convergence is usually required in this kind of nonlinear regression problem). Note that, at the predicted point (around 15–16 bits of weight precision), the squared difference curve dives. That implies the inability to converge to the desired mapping when the number of bits for the weights is less than 15 bits.

Similar supporting results are observed in the XOR classification problem using an MLP with 2 inputs, 3 hidden units, and 1 output [see Fig. 10(b)]. Due to the classification nature of the XOR problem, a soft convergence is good enough for the termination of training. Simulation results show that at the predicted point of 12–13 bits of weight precision, the squared difference curve dives. Additional low precision simulation results are reported for large scale classification problems in [12].

V. CONCLUDING REMARKS

The paper is devoted to the derivation of finite precision error analysis techniques for neural network implementations, especially analysis of the back-propagation training of MLP's. The analysis technique is versatile in that it prepares the ground for a wide variety of neural network algorithms: recurrent neural networks, competitive learning networks, etc. All of these algorithms share similar computational mechanisms to those used in back-propagation learning.

TABLE IV
RESULTS: PRECISION REQUIREMENTS IN MULTILAYER PERCEPTRONS

	Back-Propagation Learning		Forward Retrieving	
	Soft conv. (binary)	Hard conv. (regression)	Soft conv. (binary)	Hard conv. (regression)
Weights, biases, & wt. updates	13–15 bits†	15–16 bits†	7 bits	8 bits
Activation, inputs, & targets	8–9 bits	9–10 bits	7 bits	8 bits

†Fewer weight bits may be used in the forward retrieving and delta computation steps of learning.

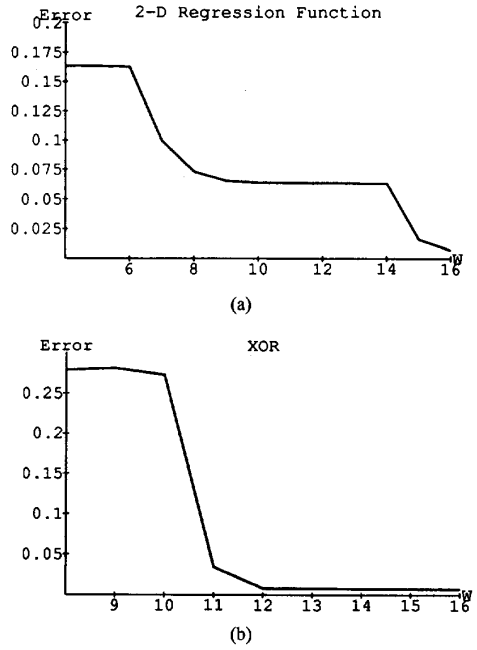

Fig. 10. The average squared differences between the desired and actual outputs of the (a) 2-D regression problem, and (b) the XOR problem after thenetworks converge.

For forward retrieving and back-propagation learning, the precisions required such that low precision computation avoids excessive diversion from the trajectory of high precision computation are summarized in Table IV.

ACKNOWLEDGMENT

The authors wish to thank Dr. D. Hammerstrom, T. Baker, Prof. Y.-H. Hu, and the anonymous reviewers of IEEE TRANSACTIONS ON COMPUTERS for their valuable and constructive suggestions, from which the revision of this paper has benefited significantly.

REFERENCES

[1] D. Hammerstrom, "A VLSI architecture for high-performance, low cost, on-chip learning," in *Proc. Int. Joint Conf. Neural Networks,* San Diego, CA, June 1990, pp. II:537–543.
[2] J. L. Holt and J. N. Hwang, "Finite precision error analysis for neural network hardware implementation," in *Proc. Int. Joint Conf. Neural Networks,*, Seattle, WA, July 1991, pp. I:519–526.
[3] S. M. Pizer with V. L. Wallace, *To Compute Numerically, Concepts and Strategies.* Boston, MA: Little, Brown and Co., 1983.
[4] J. N. Hwang, J. A. Vlontzos, and S. Y. Kung, "A systolic neural network architecture for hidden Markov models," *IEEE Trans. Acoust., Speech, Signal Processing,* vol. 37, pp. 1967–1979, Dec. 1989.
[5] ——, "A unified architecture for artificial neural networks," *J. Parallel Distributed Comput.,* pp. 358–387, Apr. 1989.
[6] A. Papoulis, *Probability, Random Variables, and Stochastic Processes.* New York: McGraw-Hill, 1984.
[7] P. J. Werbos, "Beyond regression: New tools for prediction and analysis in the behavior science," Ph.D. dissertation, Harvard Univ., Cambridge, MA, 1974.
[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing (PDP): Exploration in the Microstructure of Cognition (Vol. 1),* Cambridge, MA: M.I.T. Press, 1986, ch. 8.
[9] J. N. Hwang and P. S. Lewis, "From nonlinear optimization to neural network learning," in *Proc. 24th Asilomar Conf. Signals, Syst., & Comput.,* Pacific Grove, CA, Nov. 1990, pp. 985–989.
[10] T. E. Baker, "Implementation limits for artificial neural networks," Master thesis, Dep. Comput. Sci. and Eng., Oregon Graduate Institute of Science and Technology, 1990.
[11] P. S. Lewis and J. N. Hwang, "Recursive least squares learning algorithms for neural networks," in *Proc. SPIE's Int. Symp. Opt. and Optoelectron. Appl. Sci. and Eng.,* San Diego, CA, July 1990, pp. 28–39.
[12] J. L. Holt and T. E. Baker, "Back propagation simulations using limited precision calculations," in *Proc. Int. Joint Conf. Neural Networks,* Seattle, WA, July 1991, pp. II: 121–126.

Jordan L. Holt received the B.S. degree in electrical engineering from the California Institute of Technology in 1989 and the M.S. degree in electrical engineering from the University of Washington in 1991.

His research was in the use of low precision arithmetic for hardware implementations of neural networks. He is currently an applications engineer at Adaptive Solutions, Inc., Beaverton, OR. He is participating in the development of software for speech and speaker recognition using both template- and connectionist-based approaches.

Jenq-Neng Hwang (S'81–M'89) received the B.S. and M.S. degrees, both in electrical engineering, from the National Taiwan University, Taipei, Taiwan, in 1981 and 1983, respectively.

After two years of obligatory military services, he enrolled as a research assistant in 1985 at the Signal and Image Processing Institute, Department of Electrical Engineering, University of Southern California, where he received the Ph.D. degree in December 1988. Since 1989, he has been with the Department of Electrical Engineering, University of Washington at Seattle as an Assistant Professor. His research interests include computational neural networks, signal/image processing, parallel algorithm design, and VLSI array architecture.

Dr. Hwang served as the Secretary of the Neural Systems and Applications Committee of the IEEE Circuits and Systems Society from 1989 to 1991, and is a member of the Technical Committee on VLSI Signal Processing and a member of the Technical Committee on Neural Networks for Signal Processing of the IEEE Signal Processing Society. Currently, he is an associate editor for IEEE TRANSACTIONS ON NEURAL NETWORKS and an associate editor for IEEE TRANSACTIONS ON SIGNAL PROCESSING.