JOURNAL OF COMPUTATIONAL PHYSICS **91**, 110–131 (1990)

Neural Algorithm for Solving Differential Equations

HYUK LEE

Department of Electrical Engineering, Polytechnic Institute of New York, Brooklyn, New York 11201

AND

IN SEOK KANG

Department of Chemical Engineering, California Institute of Technology, Pasadena, California 91125

Received August 17, 1988; revised October 6, 1989

Finite difference equations are considered to solve differential equations numerically by utilizing minimization algorithms. Neural minimization algorithms for solving the finite difference equations are presented. Results of numerical simulation are described to demonstrate the method. Methods of implementing the algorithms are discussed. General features of the neural algorithms are discussed. © 1990 Academic Press, Inc.

I. INTRODUCTION

Numerical computation in many disciplines, such as physics, applied mathematics, electrical engineering, biochemistry, etc., has received a great deal of attention recently as a practical technique to understand complex phenomena that are almost impossible to treat analytically [1]. Supercomputers have been built to speed up the calculation. Furthermore, new computing algorithms based on the concept of concurrent processing have been developed and implemented by connecting a small number of processors.

Recently, highly parallel neural networks have been investigated extensively to solve complicated problems such as pattern recognition and combinatorial optimization [2]. Linear simultaneous equations also have been treated by applying neural networks [3]. Implementation of neural networks by utilizing volume holographic optical interconnections have proved to be promising [4].

One of the most general methods of solving differential equations is to use finite difference equations and to solve the algebraic equations [5]. The computational load for solving the difference equations increases very fast as the number of discrete points becomes large. Therefore, a highly parallel algorithm to solve the finite difference equations is essential when a complicated problem is encountered. In this paper, neural algorithms for minimization are utilized to develop highly parallel algorithms for solving the finite difference equations. In Section II, the basic idea of neural networks is reviewed, and general continuous or discrete neural minimization algorithms are introduced. As an example, in Section III, the differential equation $u' = f(u)$ is considered to show how the neural minimization algorithms can solve the equation, and the result of numerical simulation of the equation $u' = u$ is described. In Section IV, general continuous and discrete neural algorithms for solving a wide range of complex partial differential equations are derived. In Section V, implementation schemes of neural algorithms utilizing high-capacity optical interconnecting devices are described. Characteristic features of the neural algorithms compared with conventional algorithms are discussed in Section VI.

II. Neural Minimization Algorithms

A. *Collective Computation of Neural Networks*

Neural networks consist of individual processors and interconnections between the processors. As an example, the schematic representation of a simple neural network is drawn in Fig. 1. Each processor is called as a neuron which has an analogy in biological neural systems. Each neuron can have two different states, i.e., on and off which are represented by binary numbers 1 and 0 (or 1 and -1). The operation of all the neurons is the same. Typically, each neuron sums all the signals coming from all the other neurons through the weighted interconnections, thresholds the summed signal to 0 or 1 (or -1 or 1), and changes its state according to the thresholded output. This operation occurs at every neuron in the network simultaneously or randomly. Therefore, if the total number of neurons in the network is large, the state of the whole network consisting of the individual neuronal state changes dynamically, and the network exhibits cooperative effects. This kind of cooperative effect is very common in statistical physics.

The neural network described above can be utilized for computation. The binary states of the neurons can be identified as a binary representation of some variables. The interconnection strenghts between the neurons may represent the information

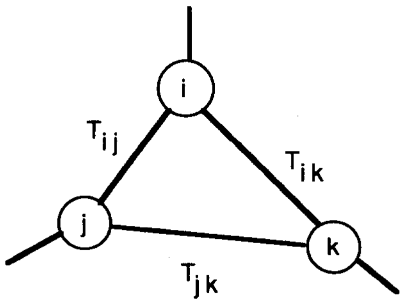

FIG. 1. Schematic diagram of a simple neural network. T_{ij}, T_{ik}, and T_{jk} are connection strengths between the pairs of neurons (i, j), (i, k), and (j, k).

of the specific problem. Then, an initial trial state for the network representing the initial trial solution may converge to the final state of the network according to the neural dynamics, and it may give the solution of the problem. This method of computation is based on the collective interaction between the neurons, and it exhibits a high degree of parallelism. Another characteristic of the neural network is that the processing of each neuron is extremely simple, however, the large number of neurons and interconnections yields enormous computational power.

B. *Continuous Neural Minimization Algorithm*

The Hopfield model has been successfully applied for solving combinatorial optimization problems such as the Hitchcock problem and the travelling salesman problem. The Hopfield model [6] is based on the nonlinear dynamic interaction between globally interconnected neurons. The states of individual neurons are specified by their outputs V_i which range between 0 and 1. The dynamics of neurons, i.e., updating rules of the neuronal states, in the Hopfield model can be summarized as follows. In the continuous model, neurons change their states according to the following equations of dynamics

$$dU_i/dt = \sum_j T_{ij} V_j, \tag{2.1}$$

$$V_i = g(U_i), \tag{2.2}$$

where t is the continuous time which corresponds to the updating parameter, T_{ij} is the interconnection strength, and $g(x)$ is a nonlinear function whose form can be taken to be

$$g(x) = (1/2)[1 + \tanh(x/x_0)], \tag{2.3}$$

which is a monotonically increasing function bounded between 0 and 1. x_0 is a constant. Hopfield has shown that if $T_{ij} = T_{ji}$, neurons in the continuous model always change their states in such a way that they minimize an energy function defined by

$$E = -(1/2) \sum_i \sum_j T_{ij} V_i V_j, \tag{2.4}$$

and stop at minima of this function. The updating rule represented by Eqs. (2.1), (2.2), and (2.3) for minimizing the energy function (2.4) is highly parallel, and it has been implemented by utilizing optics [7].

The Hopfield model described above can be applied only to minimization of quadratic energy functions which have symmetric T matrices. However, the Hopfield model for arbitrary energy functions has been investigated recently [8]. Consider an energy function for a minimization problem that can be written by

$$E = F(V_i), \tag{2.5}$$

where F is a non-singular and bounded function of variables V_i, and the partial derivatives with respect to V_i are assumed to be well defined. E is assumed to be always positive or zero, which is valid because E is bounded. The time evolution of the energy function defined by Eq. (2.5) is given by

$$dE/dt = \sum_i (\partial F/\partial V_i)(dV_i/dt). \tag{2.6}$$

If the updating rule is defined by

$$dU_i/dt = -\partial F/\partial V_i, \tag{2.7}$$

$$V_i = h(U_i), \tag{2.8}$$

where $h(x)$ is a monotonically increasing function bounded between 0 and 1, and U_i is the intermediate variable, it can be proved that Eqs. (2.7) and (2.8) minimize the energy function as follows. From Eq. (2.8), we have

$$dV_i/dt = h'(U_i)(dU_i/dt), \tag{2.9}$$

where h' is the derivative with respect to x. $h'(x)$ is always positive or zero because $h(x)$ is a monotonically increasing function. If the updating rule represented by Eqs. (2.7) and (2.8), and Eq. (2.9) is used in Eq. (2.6), we have

$$dE/dt = -\sum_i h'(U_i)(dU_i/dt)^2. \tag{2.10}$$

Equation (2.10) is always negative or zero. Therefore, the change of the energy function in time according to the updating rule Eq. (2.7) and (2.8) guarantees minimization of the energy function.

C. *Discrete Neural Minimization Algorithm*

The synchronous discrete neural algorithm for minimizing a wide range of energy functions has been developed recently [9]. The energy function is assumed to be an arbitrary type of polynomial function of the state variables. Real binary variables having values 1 and -1 are considered as state variables. The energy function can be described as

$$E = F(\{B_1, B_2, ..., B_N\}), \tag{2.11}$$

where B's are the state variables and the total number of state variables is N.

Partially synchronous minimization is considered for the most general case. Totally synchronous or totally asynchronous minimizations are specific examples of the general case. Assume that, at each step, M state variables are selected randomly, and minimization is carried out by updating the M state variables simultaneously and leaving all the other state variables unchanged. M can be any integer from 1 to N, and the minimization algorithm becomes totally asynchronous or totally synchronous if M is equal to 1 or N. A set P is defined to consist of the indices for the selected state variables. Another set $P' = \{1, 2, ..., N\} - P$ is defined to represent the indices of the state variables which are not changed. The updated state variables B'_i and the change of the state variables ΔB_i satisfy the relation

$$B'_i = B_i + \Delta B_i, \tag{2.12}$$

where $i \in P$. The updated state variables are also binary variables having values 1 and -1. Therefore, the possible values of ΔB_i are

$$\Delta B_i = -2, 0, 2. \tag{2.13}$$

The incremental change in energy ΔE due to the updated state variables given by Eq. (2.12) is considered to develop an algorithm which minimizes the energy function described by Eq. (2.11). ΔE is defined as

$$\Delta E = E[\{B'_i, B_j\}] - E[\{B_i, B_j\}], \tag{2.14}$$

where $i \in P$ and $j \in P'$, and utilizing Eq. (2.12), it becomes

$$\Delta E = E[\{B_i + \Delta B_i, B_j\}] - E[\{B_i, B_j\}]. \tag{2.15}$$

The first term in the right-hand side of Eq. (2.15) can be expanded as a Taylor's series in several variables because E is a polynomial of the state variables. Therefore, the incremental energy changes ΔE can be written as

$$\Delta E = \sum_{m=1} \sum_{i1 \in P} \cdots \sum_{im \in P} (1/m!)[\Delta B_{i1} \cdots \Delta B_{im}] \, D[B_{i1} \cdots B_{im}] \, E, \tag{2.16}$$

where $D[B_{i1} \cdots B_{im}]$ E is a partial derivative with respective to the state variables $B_{i1} \cdots B_{im}$ at $\Delta B_i = 0$ for all $i \in P$. The total number of terms in the summation of Eq. (2.16) is finite because E is a polynomial.

To reduce the products of changes in Eq. (2.16) to linear forms, the following relations are derived. For an arbitrary state variable B and a positive integer n, $(\Delta B)^n$ satisfies

$$(\Delta B)^n = (-2B)^{n-1} \Delta B, \tag{2.17}$$

which can be proved as follows. If ΔB is zero, Eq. (2.17) is satisfied automatically. If ΔB is 2, B and B' should be -1 and 1 according to Eq. (2.12). In this case, the right-hand side of Eq. (2.17) becomes the same as the left-hand side; i.e.,

$$[(-2)(-1)]^{n-1} [2] = 2^n = (\Delta B)^n. \tag{2.18}$$

On the other hand, if ΔB is -2, B and B' should be 1 and -1 according to Eq. (2.12). Therefore, the right-hand side of Eq. (2.17) becomes

$$[(-2)(1)]^{n-1} [-2] = [-2]^n, \tag{2.19}$$

which is the same as the left-hand side of Eq. (2.17) in the case of $\Delta B = -2$. Consider next a product of changes given by $A\Delta B_{i1} \cdots \Delta B_{im}$ with an arbitrary coefficient A. Assume that $m > 1$. The product can be written as

$$A\Delta B_{i1} \cdots \Delta B_{im} = |A|(1/2)[-\{S(A)\,\Delta B_{i1} - \Delta B_{i2} \cdots \Delta B_{im}\}^2 + (\Delta B_{i1})^2 + (\Delta B_{i2} \cdots \Delta B_{im})^2], \tag{2.20}$$

$$\leqslant |A|(1/2)[(\Delta B_{i1})^2 + (\Delta B_{i2} \cdots \Delta B_{im})^2], \tag{2.21}$$

where $S(A)$ is the sign of A. The first term in Eq. (2.20) is always negative or zero, and Eq. (2.21) follows. The second term in Eq. (2.21) is a product of smaller number of changes than the left term in Eq. (2.20). Therefore, this technique can be applied iteratively to reduce the product of changes in Eq. (2.20) to a sum of individual changes, and it is given by

$$A\Delta B_{i1} \cdots \Delta B_{im} \leqslant |A| \left[\sum_{j=1}^{m-1} 2^{-j}(\Delta B_{ij})^{2^j} + 2^{-(m-1)}(\Delta B_{im})^{2^{(m-1)}} \right]. \tag{2.22}$$

Equation (2.22) can be symmetrized by considering m cyclic orderings of indices $\{i1, ..., im\}$. Equation (2.22) now is given by

$$A\Delta B_{i1} \cdots \Delta B_{im} \leqslant \left(\frac{1}{m}\right) |A| \sum_{k=1}^{m} \left[\sum_{j=1}^{m-1} 2^{-j}(\Delta B_{ik})^{2^j} + 2^{-(m-1)}(\Delta B_{ik})^{2^{(m-1)}} \right]. \tag{2.23}$$

If Eq. (2.17) is utilized, Eq. (2.23) can be written as

$$A\Delta B_{i1} \cdots \Delta B_{im} \leqslant |A|\; I(m) \sum_{k=1}^{m} B_{ik}\, \Delta B_{ik}, \tag{2.24}$$

where

$$I(m) = -\left(\frac{1}{m}\right)\left[\sum_{j=1}^{m-1} 2^{(2^j - j - 1)} + 2^{(2^{(m-1)} - m)} \right]. \tag{2.25}$$

If Eqs. (2.24) and (2.25) are used in Eq. (2.16), the incremental energy change is given by

$$\Delta E \leqslant -\sum_i G_i\, \Delta B_i, \tag{2.26}$$

where

$$G_i = -\left[B_i \left\{ \sum_{m=1} \sum_{i1 \in P} \cdots \sum_{im \in P} \left(\frac{1}{m!}\right) I(m+1) \times |D[B_i B_{i1} \cdots B_{im}]E| \right\} + D[B_i]E \right]. \tag{2.27}$$

From Eq. (2.26) it is clear that if

$$G_i \Delta B_i \geqslant 0 \qquad \text{for all} \quad i \in P, \tag{2.28}$$

the incremental energy change is always negative or zero. Therefore, Eq. (2.28) updates the state variables in such a way that it minimizes the energy function in the limit of many iterations. For positive G_i, Eq. (2.28) is satisfied if ΔB_i is positive or zero. If B_i is equal to 1, B_i' should be equal to 1 because $\Delta B_i = 0$ is the only solution. However, if B_i is equal to -1, B_i' should be 1 because this makes the incremental energy change more negative than using the condition $\Delta B_i = 0$. Therefore, if G_i is positive, the updated value becomes 1 which is the same as the sign of the value G_i. If G_i is negative, the above argument can be applied to show that the updated value for B_i' becomes -1. This is the same as the sign of G_i. If G_i is zero, B_i' can have any values. Summarizing the above result, the updating rule can be written as

$$B_i' = T(G_i) \qquad \text{for all} \quad i \in P, \tag{2.29}$$

where T is a unit step function defined by $T(x) = 1$ if $x > 0$ and $T(x) = -1$ if $x \leqslant 0$. Equation (2.29) is the general discrete neural algorithm which minimizes energy functions consisting of arbitrary types of polynomials of the state variables in a partially synchronous way.

III. Case Study for Solving Differential Equations

A. *Continuous Algorithm for $u' = f(u)$*

A simple example is considered to explain how neural minimization algorithms described in Section II can be utilized to solve differential equations numerically. The differential equation is chosen as

$$du/dx = f(u), \tag{3.1}$$

where f is a polynomial function of u. The finite difference equation for Eq. (3.1) is given by

$$(U_{a+1} - U_{a-1})/2h = f(U_a), \tag{3.2}$$

where h is the mesh size for discretization, and U_a is the discrete variable corresponding to the mesh index a. For the continuous neural algorithm, a binary representation for the variable U_a may be given by

$$U_a = \sum_{s=-K}^{K} 2^{s-1} V_{as} - \{(1/2)[2^K - 1]\}, \tag{3.3}$$

where K and K' are positive integers, and V_{as} are binary variables.

The energy function for the continuous model is defined by

$$E=\sum_{a} A[(U_{a+1}-U_{a-1})/2h-f(U_a)]^2, \tag{3.4}$$

where A is a positive constant and the updating rule is derived from the time derivative

$$\begin{aligned} dE/dt=\sum_{abs} 2A[(U_{a+1}-U_{a-1})/2h-f(U_a)]\ \{(\partial U_{a+1}/\partial V_{bs})/2h \\ -(\partial U_{a-1}/\partial V_{bs})/2h-\partial f(U_a)/\partial V_{bs}\}\ (dV_{bs}/dt). \end{aligned} \tag{3.5}$$

The expression of the intermediate variables for the updating rule given by Eq. (2.7) becomes

$$\begin{aligned} dW_{bs}/dt&=-\sum_{a} 2A[(U_{a+1}-U_{a-1})/2h-f(U_a)]\ \{(\partial U_{a+1}/\partial V_{bs})/2h \\ &\quad -(\partial U_{a-1}/\partial V_{bs})/2h-\partial f(U_a)/\partial V_{bs}\} \qquad (3.6)\\ &=-2A\{[(U_b-U_{b-2})/2h-f(U_{b-1})]/2h \\ &\quad -[(U_{b+2}-U_b)/2h-f(U_{b+1})]/2h \\ &\quad -[(U_{b+1}-U_{b-1})/2h-f(U_b)]\ (df(U_b)/dU_b)\}\ (\partial U_b/\partial V_{bs}). \qquad (3.7) \end{aligned}$$

Therefore, the continuous neural agorithm for solving Eq. (3.1) consists of

$$\begin{aligned} dW_{bs}/dt= -2A\{(1/2h)^2\,(-U_{b+2}+2U_b-U_{b-2})+(1/2h)[f(U_{b+1}) \\ -f(U_{b-1})-(U_{b+1}-U_{b-1})(df(U_b)/dU_b)] \\ +f(U_b)(df(U_b)/dU_b)]\}\ (\partial U_b/\partial V_{bs}). \end{aligned} \tag{3.8}$$

$$V_{bs}=g(W_{bs}), \tag{3.9}$$

where g is the threshold function given by Eq. (2.3), and the solution is given by the V_{bs} $(t=\infty)$.

B. *Numerical Simulation*

Numerical simulation of the continuous neural algorithm described in Section III. A was carried out to explain in detail how it works. The differential equation is chosen as

$$du/dx=u. \tag{3.10}$$

The finite difference equation for Eq. (3.10) is given by

$$(U_{a+1}-U_{a-1})/2h=U_a. \tag{3.11}$$

The binary representation for the variables U_a described in Eq. (3.3) was utilized.

The energy function is given by

$$E=\sum_a A[(U_{a+1}-U_{a-1})/2h-U_a]^2, \tag{3.12}$$

and the updating rule can be read from Eqs. (3.8) and (3.9) as follows:

$$dW_{bs}/dt=-2A\{(1/2h)^2(-U_{b+2}+2U_b-U_{b-2})+U_b)\}(\partial U_b/\partial V_{bs}). \tag{3.13}$$

$$V_{bs}=g(W_{bs}). \tag{3.14}$$

Consider the differential equation (3.10) in the interval $0<x<1$. The initial value is given by $u(0)=U_0$. The unit interval is discretized with a step h. The discretized variables are $U_1, U_2, ..., U_N$, where $hN=1$. Then, the discretized form of Eq. (3.13) in time t without thresholding is given by

$$\begin{aligned}U_b(k+1)=U_b(k)+2\Delta tA\{[(1/2h)^2(U_{b+2}(k)-2U_b(k)\\+U_{b-2}(k))-U_b(k)]\},\end{aligned} \tag{3.15}$$

where $b=1, ..., N$, k denotes the number of iterations in time, and Δt is the incremental time step for the iteration. However, Eq. (3.15) contains undefined quantities U_{-1}, U_{N+1}, and U_{N+2}. The values for these variables at each iteration are obtained by considering Eq. (3.11)

$$U_{-1}(k)=U_1(k)-2hU_0, \tag{3.16}$$

$$U_{N+1}(k)=U_{N-1}(k)+2hU_N(k), \tag{3.17}$$

$$U_{N+2}(k)=U_N(k)+2h[U_{N-1}(k)+2hU_N(k)]. \tag{3.18}$$

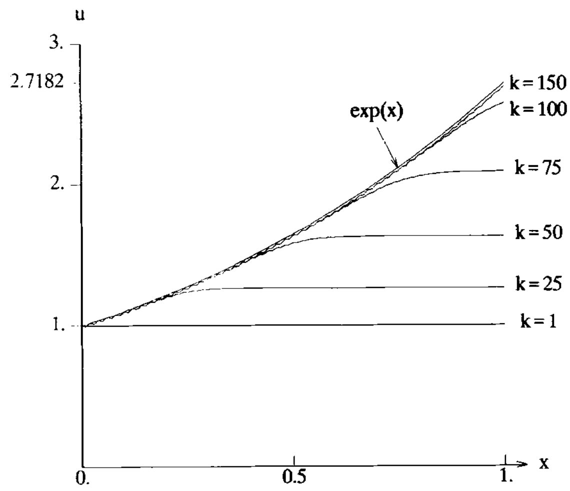

FIG. 2. Convergence of the initial trial solution $U_i=1$ for $i=1$ to 100. The solution was stabilized after 170 iterations.

Equations (3.15)–(3.18) are utilized to obtain the solution for the differential equation.

Computer simulation based on Eqs. (3.15)–(3.18) has been performed. The parameters were chosen as $h = 0.01$, $N = 100$, and $\Delta tA = 0.0001$. Here, ΔtA was chosen as a maximum value satisfying the condition that the final converged solution does not depend on ΔtA characterizing discretization of the differential equation (3.13) with respect to the parameter t. For simplicity, the algorithm without thresholding was considered, and AT&T PC6300 was used. The initial value was $U_0 = 1$. Two sets of values for U_i, $i = 1$ to N, at the $k = 0$ iteration step, i.e., initial trial solutions, were selected to show that both trial solutions converge to the correct solution. Figure 2 shows convergence of the trial solution given by $U_i = 1$ for $i = 1$ to N. Another trial solution defined by $U_i = 1.\ 5hi + 1$ for $i = 1$ to N is shown to converge to the correct solution in Fig. 3.

C. *Discrete Neural Algorithm*

Derivation of the discrete neural algorithm for solving the differential equation $u' = u$ is presented to illustrate the algorithm introduced in Section II.C. Totally asynchronous algorithm, i.e., updating one state variable at a time but randomly, is considered for simplicity. The energy function for the discrete model is given by

$$E = \sum_a [(U_{a+1} - U_{a-1})/2h - U_a]^2, \tag{3.19}$$

where we set $A = 1$. The binary representation of variables U_a is chosen as

$$U_a = \sum_{s=-K'}^{K} 2^{s-1}(B_{as} + 1)/2 - \{(1/2)[2^K - 1]\}, \tag{3.20}$$

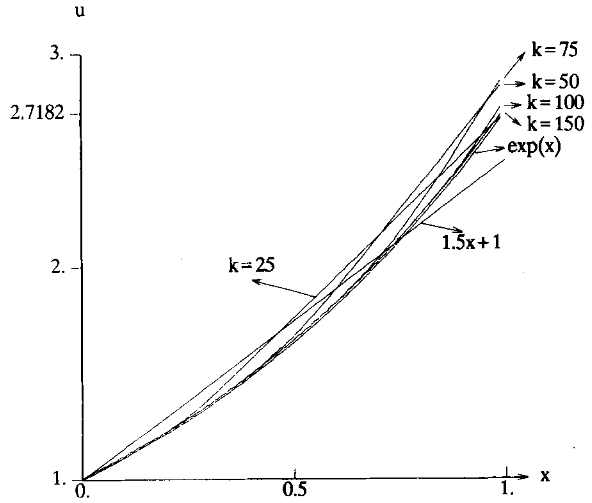

FIG. 3. Convergence of the initial trial solution $U_i = 1.5ih + 1$ for $i = 1$ to 100 and $h = 0.01$. In this case, the initial trial solution is closer to the correct solution than the initial trial solution used in Fig. 2. However, the solution was stabilized after 160 iterations.

where K and K' are positive integers, and B_{as} are binary variables having values 1 and -1. The incremental change in energy due to the variable B_{cs} is

$$\begin{aligned}\Delta E &= [(U_{c+1} - U_{c-1})/2h - U_c - 2^{n-2}\,\Delta B_{cs}]^2 \\ &\quad + [(U_c + 2^{n-2}\,\Delta B_{cs} - U_{c-2})/2h - U_{c-1}]^2 \\ &\quad + [(U_{c+2} - U_c - 2^{n-2}\,\Delta B_{cs})/2h - U_{c+1}]^2 \\ &\quad - [(U_{c+1} - U_{c-1})/2h - U_c]^2 \\ &\quad - [(U_c - U_{c-2})/2h - U_{c-1}]^2 \\ &\quad - [(U_{c+2} - U_c)/2h - U_{c+1}]^2 \\ &= 2\{-(U_{c+2} - 2U_c + U_{c-2})(1/2h)^2 + U_c\}\, 2^{n-2}\,\Delta B_{cs} \\ &\quad - 2\{1 + 2(1/2h)^2\}\,(2^{n-2})^2\, B_{cs}\,\Delta B_{cs},\end{aligned} \tag{3.21}$$

which gives the intermediate variable for the discrete model as

$$\begin{aligned}W_{cs} &= 2\{(U_{c+2} - 2U_c + U_{c-2})(1/2h)^2 - U_c\}\, 2^{n-2} \\ &\quad + 2\{1 + 2(1/2h)^2\}(2^{n-2})^2\, B_{cs}.\end{aligned} \tag{3.22}$$

Equation (3.22) together with the step thresholding function gives the discrete neural algorithm.

IV. General Formalism for Solving Differential Equations

A. *General Algorithms*

The general form of differential equations to be considered in this section is represented by

$$F_i(x_l, u_j, d_m u_k, \text{higher-order derivatives}) = 0, \tag{4.1}$$

where l, $m = 1$ to M, x_1 are independent real variables, and $i, j, k = 1$ to N, u_j are dependent real variables, and $d_m u_k$ are first-order partial derivatives, and there are N functions indexed by i. The functions F_i are assumed to be non-singular, and partial derivatives with respect to x_l, u_j, $d_m u_k$, and higher-order derivatives are assumed to be well defined. The finite difference method is introduced to solve the differential equation (4.1). The finite domain in M dimensional space for independent variables is discretized with a unit volume h^M. A discrete index vector $\mathbf{a}$ with components $\mathbf{a}(l)$, $l = 1$ to M, is introduced to describe such a space. Each component $\mathbf{a}(1)$ represents the discrete coordinate for the individual variable x_l.

Then, the discretized quantities corresponding to the continuous quantities can be written as

$$x_{l\mathbf{a}} = x_{l0} + \mathbf{a}(l)h, \tag{4.2}$$

$$U_{j\mathbf{a}} = u_j(X_l(\mathbf{a})), \tag{4.3}$$

$$\begin{aligned} D_m U_{k\mathbf{a}} &= (1/2h)(U_{k,\mathbf{a}+\mathbf{m}'} - U_{k,\mathbf{a}-\mathbf{m}'}) \\ &= (1/2h)\sum_{\mathbf{b}} (\delta_{\mathbf{a}+\mathbf{m}',\mathbf{b}} - \delta_{\mathbf{a}-\mathbf{m}',\mathbf{b}})\, U_{k\mathbf{b}}, \end{aligned} \tag{4.4}$$

$$\begin{aligned} D_{lm} U_{k\mathbf{a}} &= (D_1(D_m U_k))_{\mathbf{a}} \\ &= D_1((U_{k,\mathbf{a}+\mathbf{m}'} - U_{k,\mathbf{a}-\mathbf{m}'})/2h) \\ &= (1/2h)^2 \sum_{\mathbf{b}} \{\delta_{\mathbf{a}+\mathbf{m}'+\mathbf{l}',\mathbf{b}} - \delta_{\mathbf{a}+\mathbf{m}'-\mathbf{l}',\mathbf{b}} \\ &\quad - \delta_{\mathbf{a}-\mathbf{m}'+\mathbf{l}',\mathbf{b}} + \delta_{\mathbf{a}-\mathbf{m}'-\mathbf{l}',\mathbf{b}}\}\, U_{k\mathbf{b}}, \end{aligned} \tag{4.5}$$

where δ is a unit matrix, x_{l0} is the initial value for th coordinate x_l, $\mathbf{m}'$, and $\mathbf{l}'$ are M-dimensional vectors satisfying the conditions $\mathbf{m}'(m) = 1$ and $\mathbf{m}' = 0$ otherwise and $\mathbf{l}'(l) = 1$ and $\mathbf{l}' = 0$ otherwise. The finite difference equation for the differential equation (4.1) is then given by

$$\begin{aligned} &F_{i\mathbf{a}}(X_{l\mathbf{a}}, U_{j\mathbf{a}}, D_m U_{k\mathbf{a}}, \text{higher-order discrete derivatives}) \\ &\quad = F'_{i\mathbf{a}}(U_{j\mathbf{b}}) = 0. \end{aligned} \tag{4.6}$$

Continuous algorithm is considered first. The variables $U_{j\mathbf{b}}$ are continuous real variables. However, we may use different types of number representation of the real variable. Let one of the possible representations of the real number be given by

$$U_{j\mathbf{b}} = R(B_{j\mathbf{b}r}), \tag{4.7}$$

where R is the representation function, r denotes the bits, and $B_{j\mathbf{b}r}$ are bit variables having two binary values 0 and 1. If Eq. (4.7) is used, the finite difference equation (4.6) becomes

$$\begin{aligned} &F_{i\mathbf{a}}(X_{l\mathbf{a}}, U_{j\mathbf{a}}, D_m U_{k\mathbf{a}}, \text{higher-order discrete derivatives}) \\ &\quad = F'_{i\mathbf{a}}(U_{j\mathbf{b}}) = F''_{i\mathbf{a}}(B_{j\mathbf{b}r}) = 0. \end{aligned} \tag{4.8}$$

To exploit a continuous algorithm, continuous variables $V_{j\mathbf{b}r}$ which are bounded between 0 and 1 are substituted with $B_{j\mathbf{b}r}$ in Eq. (4.8). Then, Eqs. (4.7) and (4.8) become

$$U_{j\mathbf{b}} = R(V_{j\mathbf{b}r}), \tag{4.9}$$

$$\begin{aligned} &F_{i\mathbf{a}}(X_{l\mathbf{a}}, U_{j\mathbf{a}}, D_m U_{k\mathbf{a}}, \text{higher-order discrete derivatives}) \\ &\quad = F'_{i\mathbf{a}}(U_{j\mathbf{b}}) = F''_{i\mathbf{a}}(V_{j\mathbf{b}r}) = 0. \end{aligned} \tag{4.10}$$

Equation (4.10) can be solved by utilizing a minimization algorithm described in II.A. An energy function for Eq. (4.10) is introduced as

$$E(V_{j\mathbf{b}r}) = A \sum_{i\mathbf{a}} F''^2_{i\mathbf{a}}, \tag{4.11}$$

where A is a positive constant. If the energy function given by Eq. (4.11) is minimized with respect to $V_{j\mathbf{b}r}$, the solution of the differential equation is given by the values $U_{j\mathbf{b}}(\infty)$ which minimize Eq. (4.11). The time derivative of Eq. (4.11) is given by

$$dE(V_{j\mathbf{b}r}(t))/dt = \sum_{i\mathbf{a}s} (\partial E/\partial V_{i\mathbf{a}s})(dV_{i\mathbf{a}s}/dt). \tag{4.12}$$

From Eqs. (2.7) and (2.8), the minimization algorithm is given by

$$dW_{i\mathbf{a}s}/dt = -\partial E/\partial V_{i\mathbf{a}s}, \tag{4.13}$$

$$V_{i\mathbf{a}s} = G(W_{i\mathbf{a}s}), \tag{4.14}$$

where $W_{i\mathbf{a}s}$ are intermediate variables, and G is a monotonically increasing thresholding function bounded between 0 and 1.

Consider the right-hand side of Eq. (4.13). It can be given by

$$\partial E/\partial V_{i\mathbf{a}s} = A \sum_{k\mathbf{c}} (\partial F''^2_{k\mathbf{c}}/\partial V_{i\mathbf{a}s}), \tag{4.15}$$

and, from Eq. (4.10),

$$\partial F''^2_{k\mathbf{c}}/\partial V_{i\mathbf{a}s} = \sum_{j\mathbf{b}} (\partial F'^2_{k\mathbf{c}}/\partial U_{j\mathbf{b}})(\partial U_{j\mathbf{b}}/\partial V_{i\mathbf{a}s}), \tag{4.16}$$

and

$$\begin{aligned}(\partial F'^2_{k\mathbf{c}}/\partial U_{j\mathbf{b}}) = {} & \partial F^2_{k\mathbf{c}}/\partial U_{j\mathbf{b}} \\ & + \sum_{mp} [\partial F^2_{k\mathbf{c}}/\partial(D_m U_{p\mathbf{c}})](\partial(D_m U_{p\mathbf{c}})/(\partial U_{j\mathbf{b}}] \\ & + \sum_{mlp} [\partial F^2_{k\mathbf{c}}/\partial(D_{ml} U_{p\mathbf{c}})][\partial(D_{ml} U_{p\mathbf{c}})/\partial U_{j\mathbf{b}}] \\ & + \text{higher-order terms.}\end{aligned} \tag{4.17}$$

If Eqs. (4.4) and (4.5) are used, we have

$$\begin{aligned}\partial(D_m U_{p\mathbf{c}})/\partial U_{j\mathbf{b}} &= (1/2h) \sum_{\mathbf{e}} (\delta_{\mathbf{c}+\mathbf{m}',\mathbf{e}} - \delta_{\mathbf{c}-\mathbf{m}',\mathbf{e}})(\partial U_{p\mathbf{e}}/\partial U_{j\mathbf{b}}) \\ &= (1/2h)\, \delta_{jp}(\delta_{\mathbf{c}+\mathbf{m}',\mathbf{b}} - \delta_{\mathbf{c}-\mathbf{m}',\mathbf{b}})\end{aligned} \tag{4.18}$$

and

$$
\begin{aligned}
&\partial(D_{ml}U_{pc})/\partial U_{j\mathbf{b}} \\
&\quad = (1/2h)^2 \sum_{\mathbf{e}} \{\delta_{\mathbf{c}+\mathbf{m}'+\mathbf{l}',\mathbf{e}} - \delta_{\mathbf{c}+\mathbf{m}'-\mathbf{l}',\mathbf{e}} \\
&\qquad - \delta_{\mathbf{c}-\mathbf{m}'+\mathbf{l}',\mathbf{e}} + \delta_{\mathbf{c}-\mathbf{m}'-\mathbf{l}',\mathbf{e}}\}(\partial U_{p\mathbf{e}}/\partial U_{j\mathbf{b}}) \\
&\quad = (1/2h)^2\, \delta_{jp}\{\delta_{\mathbf{c}+\mathbf{m}'+\mathbf{l}',\mathbf{b}} - \delta_{\mathbf{c}+\mathbf{m}'-\mathbf{l}',\mathbf{b}} \\
&\qquad - \delta_{\mathbf{c}-\mathbf{m}'+\mathbf{l}',\mathbf{b}} + \delta_{\mathbf{c}-\mathbf{m}'-\mathbf{l}',\mathbf{b}}\}.
\end{aligned} \tag{4.19}
$$

Substituting Eqs. (4.18) and (4.19) into Eq. (4.17), it becomes

$$
\begin{aligned}
(\partial F'^2_{k\mathbf{c}}/\partial U_{j\mathbf{b}}) &= \partial F^2_{k\mathbf{c}}/\partial U_{j\mathbf{b}} \\
&\quad + \sum_{mp} [\partial F^2_{k\mathbf{c}}/\partial(D_m U_{p\mathbf{c}})](1/2h)\, \delta_{jp}(\delta_{\mathbf{c}+\mathbf{m}',\mathbf{b}} - \delta_{\mathbf{c}-\mathbf{m}',\mathbf{b}}) \\
&\quad + \sum_{mlp} [\partial F^2_{k\mathbf{c}}/\partial(D_{ml} U_{p\mathbf{c}})](1/2h)^2\, \delta_{jp}\{\delta_{\mathbf{c}+\mathbf{m}'+\mathbf{l}',\mathbf{b}} \\
&\quad - \delta_{\mathbf{c}+\mathbf{m}'-\mathbf{l}',\mathbf{b}} - \delta_{\mathbf{c}-\mathbf{m}'+\mathbf{l}',\mathbf{b}} + \delta_{\mathbf{c}-\mathbf{m}'-\mathbf{l}',\mathbf{b}}\} \\
&\quad + \text{higher-order terms} \qquad (4.20) \\
&= \partial F^2_{k\mathbf{c}}/\partial U_{j\mathbf{b}} \\
&\quad + \sum_{m} [\partial F^2_{k\mathbf{c}}/\partial(D_m U_{j\mathbf{c}})](1/2h)(\delta_{\mathbf{c}+\mathbf{m}',\mathbf{b}} - \delta_{\mathbf{c}-\mathbf{m}',\mathbf{b}}) \\
&\quad + \sum_{ml} [\partial F^2_{k\mathbf{c}}/\partial(D_{ml} U_{j\mathbf{c}})](1/2h)^2\, \{\delta_{\mathbf{c}+\mathbf{m}'+\mathbf{l}',\mathbf{b}} \\
&\quad - \delta_{\mathbf{c}+\mathbf{m}'-\mathbf{l}',\mathbf{b}} - \delta_{\mathbf{c}-\mathbf{m}'+\mathbf{l}',\mathbf{b}} + \delta_{\mathbf{c}-\mathbf{m}'-\mathbf{l}',\mathbf{b}}\} \\
&\quad + \text{higher-order terms.}
\end{aligned} \tag{4.21}
$$

From Eqs. (4.16) and (4.21), Eq. (4.15) becomes

$$
\begin{aligned}
\partial E/\partial V_{ia s} = A \sum_{j\mathbf{b}k\mathbf{c}} \Big\{ &\partial F^2_{k\mathbf{c}}/\partial U_{j\mathbf{b}} \\
&+ \sum_{m} [\partial F^2_{k\mathbf{c}}/\partial(D_m U_{j\mathbf{c}})](1/2h)(\delta_{\mathbf{c}+\mathbf{m}',\mathbf{b}} - \delta_{\mathbf{c}-\mathbf{m}',\mathbf{b}}) \\
&+ \sum_{ml} [\partial F^2_{k\mathbf{c}}/\partial(D_{ml} U_{j\mathbf{c}})](1/2h)^2\, (\delta_{\mathbf{c}+\mathbf{m}'+\mathbf{l}',\mathbf{b}} \\
&- \delta_{\mathbf{c}+\mathbf{m}'-\mathbf{l}',\mathbf{b}} - \delta_{\mathbf{c}-\mathbf{m}'+\mathbf{l}',\mathbf{b}} + \delta_{\mathbf{c}-\mathbf{m}'-\mathbf{l}',\mathbf{b}}) \\
&+ \text{higher-order terms} \Big\} (\partial U_{j\mathbf{b}}/\partial V_{ia s})
\end{aligned} \tag{4.22}
$$

$$
\begin{aligned}
&= A\sum_{kc}\Big\{\partial F_{kc}^2/\partial U_{i\mathbf{a}} \\
&\quad + \sum_{m}(1/2h)(\delta_{\mathbf{a}-\mathbf{m}',\mathbf{c}}-\delta_{\mathbf{a}+\mathbf{m}',\mathbf{c}})[\partial F_{k\mathbf{c}}^2/\partial(D_m U_{i\mathbf{c}})] \\
&\quad + \sum_{ml}(1/2h)^2(\delta_{\mathbf{a}-\mathbf{m}'-\mathbf{l}',\mathbf{c}}-\delta_{\mathbf{a}-\mathbf{m}'+\mathbf{l}',\mathbf{c}} \\
&\quad - \delta_{\mathbf{a}+\mathbf{m}'-\mathbf{l}',\mathbf{c}}+\delta_{\mathbf{a}+\mathbf{m}'+\mathbf{l}',\mathbf{c}})[\partial F_{k\mathbf{c}}^2/\partial(D_{ml}U_{i\mathbf{c}})] \\
&\quad + \text{higher-order terms}\Big\}(\partial U_{i\mathbf{a}}/\partial V_{i\mathbf{a}s}), \qquad (4.23)
\end{aligned}
$$

where Eq. (4.9) was used to sum over j and $\mathbf{b}$, and $\mathbf{a}$ and $\mathbf{c}$ in the unit matrices δ were rearranged. Utilizing the relations (4.4) and (4.5) in Eq. (4.23), the algorithm for solving the differential equation is finally given by

$$
\begin{aligned}
dW_{i\mathbf{a}s}/dt = -A\sum_{k}\Big\{&\partial(F_{k\mathbf{a}}^2)/\partial U_{i\mathbf{a}} \\
&-\sum_{m} D_m[\partial(F_{k\mathbf{a}}^2)/\partial(D_m U_{i\mathbf{a}})] \\
&+\sum_{ml} D_{ml}[\partial(F_{k\mathbf{a}}^2)/\partial(D_{ml}U_{i\mathbf{a}})] \\
&+\text{higher-order terms}\Big\}(\partial U_{i\mathbf{a}}/\partial V_{i\mathbf{a}s}), \qquad (4.24)
\end{aligned}
$$

and

$$V_{i\mathbf{a}m} = G(W_{i\mathbf{a}m}). \qquad (4.25)$$

Discrete formalism for solving differential equations is based on the discrete neural minimization algorithm developed in Section II.C. The finite difference equation for the differential equation (4.1) is given by Eq. (4.6). However, the binary representation of the variables $U_{i\mathbf{a}}$ is achieved by using binary variables having values 1 and -1. For example, Eq. (3.20) can be used. Therefore, the energy function for the discrete algorithm is given by

$$E(B_{j\mathbf{b}s}) = \sum_{i\mathbf{a}} F''_{i\mathbf{a}}(B_{j\mathbf{b}s})^2, \qquad (4.26)$$

where $B_{j\mathbf{b}n}$ are binary variables having values 1 and -1. The discrete algorithm can be obtained by using Eqs. (2.27) and (2.29).

B. *Boundary Conditions*

Dynamical equations, i.e., updating rule, which yield solutions for differential equations have been developed. However, the initial or boundary conditions of the differential equations have not been considered. In this section, the method of incorporating the initial or boundary conditions in the updating rule is discussed.

First, it is shown that initial conditions can be considered as boundary conditions. As described in Section III.B, consider a first-order differential equation of one independent variable x. The initial condition may be written as $u(x_0)=u_0$, where u is the dependent variable and x_0 is the initial point. In this case, the final point x_1 should be specified to solve the differential equation numerically. Therefore, if 0 and $N+1$, which correspond to discretized coordinates of x_0 and x_1+h, are selected as the boundary points, the variables of the problem are $U(1)$, $U(2)$, ..., $U(N)$ and the boundary conditions are given by $U(0)$ and $U(N+1)$. In this case, $U(N+1)$ is not a fixed value but varies subject to the original finite difference equation. Therefore, $U(N+1)$ can be written as a function of $U(0)$, $U(1)$, ..., $U(N)$.

Boundary conditions including initial conditions can be incorporated as follows. First, select the boundary for a problem. Then, the domain surrounded by the boundary is discretized. The boundary values are assigned to the discrete variables for the boundary points. However, the dynamical equations presented in Section IV.A are chosen only for the variables inside the boundary. The values for the boundary variables are supplied by utilizing the finite difference equations at the boundary at each iteration.

V. Implementation of Neural Algorithms

In this section, possible implementation schemes of the neural algorithms are presented. As discussed in Section II.A, one of the most important characteristics in the neural network is the interconnection between neurons, because the individual neuron performs a simple operation. Therefore, the collective computational properties emerge as a result of large number of neurons and interconnections between the neurons. Electrical implementation of a large number of interconnections, for example, 10^{12} interconnections,, is very difficult. However, optical implementation is very attractive to overcome the difficulties encountered in electrical implementation.

Optical implementation of the neural algorithms for solving differential equations is based on interpreting the algorithms equations (3.8) and (3.22) as follows. Consider a two-dimensional optical wave propagating in free space. At each iteration, the discretized independent variable V_{as} (continuous model) or B_{as} (discrete model) is represented by the light amplitude in a two-dimensional optical wave at each resolution point which is identified as a neuron. Then, at each neuron, the intermediate variable $W_{bs'}$ is obtained by simply multiplying and summing V_{as} (or B_{as}) from other neurons, i.e., neurons interconnected to the neuron represented by bs', according to Eqs. (3.8) or (3.22). Here, interconnections are achieved by optical volume holograms. Multiplication and summation can be implemented optically or electronically. The updated values for the neurons are obtained by thresholding the intermediate variables. Thresholding can be performed by using nonlinear optical phenomena [10] or an electronic method. Finally, the updated values for the neurons represented by a two-dimensional optical wave are fed back to the incident port of the optical system. Gain is included in the feedback loop to compensate the loss. The solution of the differential equation is obtained when the iteration converges. The above scheme of optical implementation is illustrated in Fig. 4.

Optical implementation of the interconnection based on volume holography utilizing photorefractive crystals has been extensively investigated [4]. The motivation for using volume holograms comes from their ability to store information in three dimensions. Using volume holographic technique, it is possible to impress a grating pattern into such a crystal so as to transfer a light signal from an input point to another point in an output plane. Another input–output connection can similarly be implemented by a second grating oriented inside the crystal at a different angle from the first one. The maximum number of connections that can be specified if a volume hologram is used is upperbounded by the degrees of freedom available in the volume of the crystal which is equal to V_H/λ^3, where V_H is the volume of the hologram and λ is the optical wave length. As a numerical example, consider an optical wave with wavelength $1\mu m$, a 1 cm square input plane. and 1 cm cube volume holograms. Then, the total number of neurons which can be processed in the system is 10^8, and the total number of interconnections becomes 10^{12}.

The proposed optical system fully exploits the advantages of optical parallel computing. It makes use of a two-dimensional optical wave to represent discretized independent variables. High-capacity volume holograms are used to interconnect a huge number of variables. Furthermore, the system is one of the most parallel systems that can be implemented to solve the finite difference equations. The total

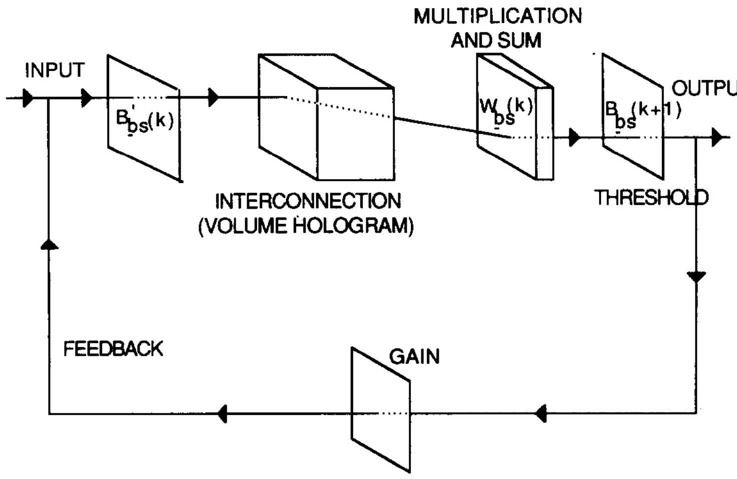

FIG. 4. Optical system for solving differential equations. Mirrors and beam splitters for the optical wave are not drawn.

number of interconnections may be reduced in actual implementation due to the Bragg cone effect, higher-order cross-talk effects, and material quality. However, 10^{10} interconnections seem to be feasible.

VI. Discussions

The main purpose of this paper is to introduce the concept of neural networks and to develop neural algorithms for solving differential equations. The detailed analysis on the algorithms will be published separately. However, general features of the neural algorithms will be discussed in this section. The remarkable collective computational properties such as recognition from partial input, robustness, and error-correction capability have been demonstrated recently. As an example, Hopfield introduced associative memory in the neural network described in Section II.A. The memory vectors $B_i^{(m)}$, where i represents the neurons and m describes the number of memory vectors, are stored in the weight of the interconnections as $T_{ij} = \sum B_i^{(m)} B_j^{(m)}$ summed over the memory vectors. The computational function of the Hopfield neural network is to reconstruct the whole original memory given an initial partial memory. In simulations, correct convergence was obtained for the total number of four memory vectors in the case of 30 neurons when the initial trial memory vector differed from the original memory vector by seven bits [6]. Therefore, the radius of convergence of the neural network is in general very large. This property is very important in solving differential equations. In conventional algorithms, a good initial guess of the solution is essential in solving differential equations effectively. However, the radius of convergence in neural algorithms is expected to be very large, and thus, selection of an initial trial solution is not so sensitive. This has been demonstrated in the results of numerical simulation shown in Figs. 2 and 3.

Another important characteristic of the neural network is nonlinear dynamics introduced by nonlinear thresholding. This property is expected to yield a large processing gain in neural computation. As an example, for the neural associative memory described above in this section, the number of iterations which yielded the correct original memory was approximately 5. Therefore, it is expected that the rate of convergence in neural algorithms is much faster than the conventional algorithms. The second property of neural algorithms which affects the rate of convergence is the collective processing inherent in the neural networks. This is because the greater the problem size, the more neurons participate collectively in solving the differential equations. To illustrate this, consider a stiff differential equation. To solve this equation, the number of mesh points should be very large so that discretization of the equation is accurate and smooth. This means the total time, i.e., the rate of convergence, required for solving the equation increases rapidly if the conventional serial algorithms are used. In the neural algorithms proposed in this paper, the equation can be solved by increasing the total number of neurons. However, the time for solving the equation actually decreases even if the total number of neurons increases, because at each iteration all the neurons interact with each other in parallel, i.e., collectively. This point has been demonstrated by considering the specific example described in Fig. 2. We solved the same problem assuming all the same conditions except decreasing the total number of mesh points between 0 and 1. Fifty mesh points instead of 100 mesh points were chosen, and h was 0.02 in this case. The constant ΔtA was 0.0001. The result of the numerical simulation is shown in Fig. 5. Comparing Figs. 2 and 5, it is clear that as the number of neurons increase, the rate of convergence decreases. Specifically, the rate of convergence for this example decreased by half when the number of neurons increased twice. The third property which decreases the time for solving differential equations is the optical feedback used in implementing the neural algorithms. This means communication between processors, i.e., neurons, is performed by the fastest physical method. Therefore, optical implementation reduces the computation time even if the total number of iterations becomes very large.

In the above paragraph, the rate of convergence for the stiff differential equations has been investigated by changing the total number of mesh points, i.e., neurons. However, it is also important to study the rate of convergence for the stiff differential equations maintaining the same total number of mesh points and changing the degree of stiffness. As an example, a differential equation $du/dx = -\lambda u$ was investigated in the interval $0 \leqslant x \leqslant 1$ with initial condition $u(0) = 1$, where λ is a positive constant and serves as a stiffness parameter. The updating rule for this case can be obtained from Eq. (3.8),

$$U_b(k+1) = U_b(k) + 2\Delta tA\{[(1/2h)^2 (U_{b+2}(k) - 2U_b(k) + U_{b-2}(k)) - \lambda^2 U_b(k)]\}, \tag{6.1}$$

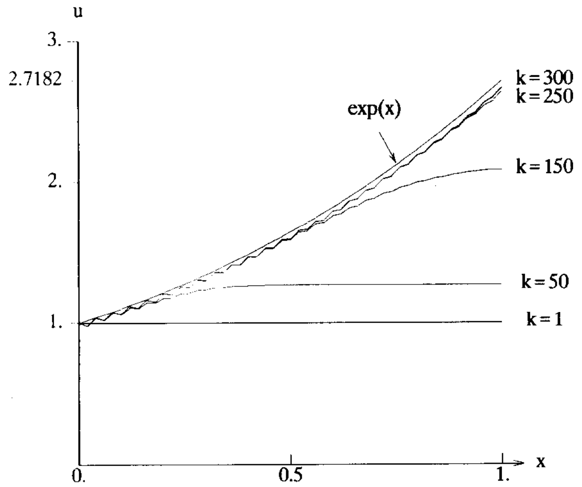

FIG. 5. Convergence of the same problem as described in Fig. 2 when the number of mesh points was decreased to 50.

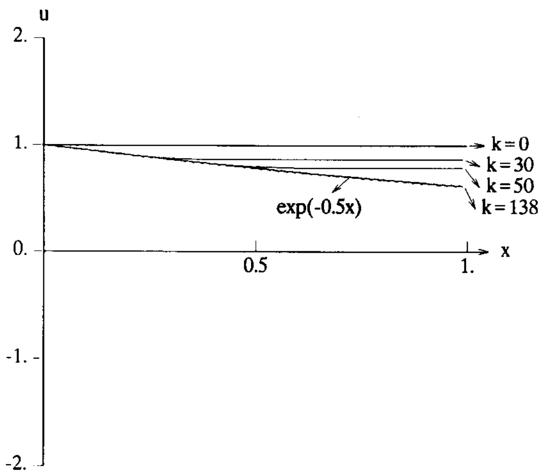

FIG. 6. Convergence of the differential equation $u' = -0.5u$. The converged solution was obtained at $k = 138$.

which is the same as Eq. (3.15) except for the λ^2 factor. The boundary conditions are given by

$$U_{-1}(k) = U_1(k) + 2h\lambda U_0, \tag{6.2}$$

$$U_{N+1}(k) = U_{N-1}(k) - 2h\lambda U_N(k), \tag{6.3}$$

$$U_{N+2}(k) = U_N(k) - 2h\lambda[U_{N-1}(k) - 2h\lambda U_N(k)]. \tag{6.4}$$

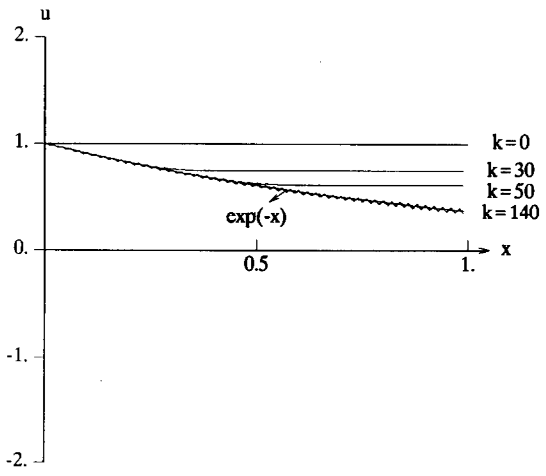

FIG. 7. Convergence of the differential equation $u' = -u$. The converged solution is not good compared with Fig. 6.

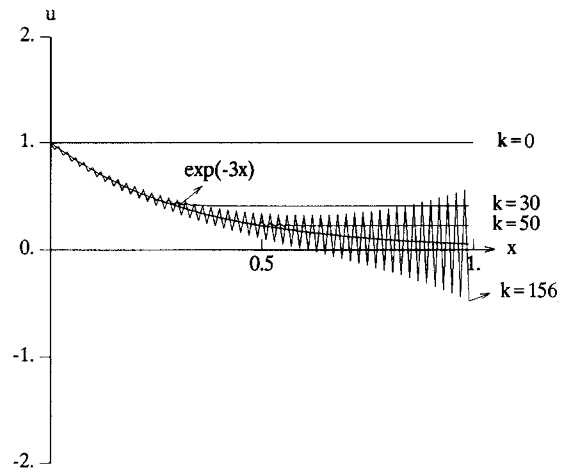

FIG. 8. Convergence of the differential equation $u' = -3u$. The converged solution was obtained at $k = 156$. However, the solution looks very bad compared with Figs. 6 and 7.

Computer simulations based on Eqs. (6.1)–(6.4) have been carried out for three different values of $\lambda = 0.5$, 1, and 3. The other parameters were chosen as $h = 0.01$, $N = 100$, and $\Delta t A = 0.0001$ for the three different values of λ, which are the same parameters as those used in Fig. 2. The same initial trial solution for three different values of λ were chosen as $U_i(0) = 1$, $i = 1, \ldots, 100$. The numerical results are shown in Figs. 6–8. Figure 6 represents the case of $\lambda = 0.5$, and the algorithm converged at $k = 138$. Figure 7 shows the result of $\lambda = 1$. The solution in this case converged at $k = 140$. However, as shown in the figure the converged solution is not good compared with the case of $\lambda = 0.5$. When the stiffness parameter λ was increased to 3, the algorithm converged at $k = 156$ as shown in Fig. 8. However, this solution is much worse than the case of $\lambda = 1$. From this simple example, we see that the rate of convergence and effectiveness of the proposed neural algorithms deteriorates when the differential equations become stiffer and the total number of mesh points, i.e., neurons, are kept the same.

Finally, the neural algorithms proposed in this paper have several advantages over the conventional parallel algorithms. First, the neural algorithms are much simpler than others. The only algebraic operations required in the neural algorithms are addition and multiplication. Inverse or other complicated logic operations are not needed. Second, the neural algorithms are the most parallel, compared with conventional parallel algorithms. For example, the algorithms for a hypercube have limited parallelism due to a small number of interconnections. Simplicity in implementation of the neural algorithms is the third advantage.

VII. SUMMARY

Highly parallel neural algorithms for solving finite difference equations have been developed. They can be applied to a wide range of differential equations. Nonlinear

thresholding which is a characteristic of neural networks has been introduced in the algorithms. Implementation of the algorithms can be realized by using electronic processors if the order of the differential equations is low, because in this case the number of interconnections is very small. However, if higher-order differential equations are considered, optical implementation is better. Finally, general features of the neural algorithms for solving differential equations have been discussed.

Acknowledgments

The authors acknowledge valuable comments by the reviewers. H. Lee would like to acknowledge support by the National Science Foundation Grant No. EET-8810288 and the Center for Advanced Technology in Telecommunications. We thank W. S. Baek for computer simulation.

References

1. *Physics Today, Special Issue: Computational Physics,* October (1987).
2. J. J. Hopfield and D. W. Tank, *Biol. Cybernet.* **52**, 141 (1985).
3. M. Takeda and J. W. Goodman, *Appl. Opt.* **25**, 3033 (1986).
4. H. Lee, X. Gu, and D. Psaltis, *J. Appl. Phys.* **65**, 2191 (1989).
5. G. F. Garrier and C. E. Pearson, *Partial Differential Equations: Theory and Technique* (Academic Press, New York, 1976).
6. J. J. Hopfield, *Proc. Nat. Acad. Sci.* **79**, 2554 (1982).
7. D. Psaltis and N. Farhat, *Opt. Lett.* **10**, 98 (1985).
8. C. Koch, J. Marroquin, and A. Yuille, *Proc. Nat. Acad. Sci.* **83**, 4263 (1986).
9. H. Lee, *Technical Digest of the Topical Meeting on Optical Computing, Salt lake city, Utah, Feb. 27–Mar. 1,* 1989.
10. H. M. Gibbs, *Optical Bistability: Controlling Light with Light* (Academic Press, San Diego, 1985).