Pergamon

Neural Networks, Vol. 8, No. 2, pp. 251–259, 1995

0893-6080/95 $9.50 + .00

0893-6080(94)00084-0

CONTRIBUTED ARTICLE

A Learning Rule of Neural Networks via Simultaneous Perturbation and Its Hardware Implementation

YUTAKA MAEDA, HIROAKI HIRANO, AND YAKICHI KANATA

Kansai University

(*Received* 6 *May* 1993; *revised and accepted* 29 *August* 1994)

Abstract—*This paper describes a learning rule of neural networks via a simultaneous perturbation and an analog feedforward neural network circuit using the learning rule. The learning rule used here is a stochastic gradient-like algorithm via a simultaneous perturbation. The learning rule requires only forward operations of the neural network. Therefore, it is suitable for hardware implementation. First, we state the learning rule and show some computer simulation results of the learning rule. A comparison between the learning rule, the usual back-propagation method, and a learning rule by a difference approximation is considered through the exclusive-OR problem and a simple pattern recognition problem known as the TCLX problem. Moreover, 26 alphabetical characters' recognition is handled to confirm a feasibility of the learning rule for large neural networks. Next, we describe details of the fabricated neural network circuit with learning ability. The exclusive-OR problem and the TCLX problem are considered. In a fabricated analog neural network circuit, input, output, and weights are realized by voltages.*

Keywords—Analog feedforward neural network circuit, Simultaneous perturbation, Learning rule, Hardware implementation.

1. INTRODUCTION

Nowadays, we can implement artificial neural networks using several media (De Gloria, 1989; Mead & Ismail, 1989). In such implementations, emulation by a digital computer is the most widely used. The high-speed emulation of neural networks becomes possible by the advance in digital computer technology. Typical examples are digital processors designed to emulate neural networks (e.g., Neuro-07 by NEC or NEUROSIM/L by Fujitsu). These approaches for neural networks' implementation will be developed much more. At the same time, we also know that digital computer may not be the most adequate medium for neural networks' implementation.

Implementation by hardware elements (e.g., electronic element or optical elements, etc.) is profitable because of parallel operations, even though such a neural network is not flexible with respect to structure (number of cells, number of layers or connections between cells, etc.) and a learning rule. A hardware realization of neural network is very important in that it operates very fast. To apply neural networks to various areas, realizing the neural network physically is an imperative issue.

Requests for reprints should be sent to Yutaka Maeda, Department of Electrical Engineering, Kansai University, 3-3-35, Yamatecho, Suita, Osaka 564 Japan; E-mail: maedayuta@kansai-u.ac.jp

We consider an analog hardware implementation of feedforward neural networks with learning ability. One of the difficulties of implementing such neural networks by physical elements is realization of its learning rule. Usually, the back-propagation method (Rumelhart et al., 1986) is the most widely used as a leaning rule of neural networks in software emulation. To use the back-propagation method, we must calculate the first-differential coefficient of the error function corresponding to all weights. The calculation of this first derivative consists of multiplications, additions, and the sigmoid functions. Then, it is complicated to embody this calculation, electrically. Moreover, if the circuit gets intricate, we must take a dynamic range of the weights and the offset errors into account more crucially (Eberhardt et al., 1992).

Such being the cases, the difficulties of implementing neural networks with learning ability depend on the learning rule. Accordingly, we must contrive a suitable learning rule for the hardware implementation.

From this point of view, we propose a learning rule via a simultaneous perturbation. In this learning rule, we substitute a mechanism calculating first-differential coefficient by a kind of a difference approximation of the error function. By using the value of the error function with the perturbation and the value of the error function without the perturbation, we can apply a kind of a difference approximation to approximate the first-

differential coefficient. In this case, only forward operations of the neural network give the modified quantities of each weight. Therefore, configuration of the circuit becomes simple.

Usually, the learning rule of neural networks via a simple sequential parameter perturbation was proposed and a hardware implementation was reported (Jabri & Flower, 1992). Independently, the authors also proposed and fabricated an analog neural network circuit using the same learning rule (Maeda, Yamashita, & Kanata, 1991) and investigated a usefulness of this type of learning rule in an inverse problem (Maeda, 1992). However, as pointed out in Maeda, Yamashita, and Kanata (1991), the learning rule using the simple perturbation requires n-times[1] forward operations of the neural network for one modification of all weights. Therefore, if the neural network is too large, it is practically impossible to expect a feasibility of this learning rule in the sense of the operation speed. Therefore, it is imperative to devise a learning rule using a simultaneous perturbation-like method. In this simple perturbation technique, perturbations are added sequentially to all weights. Cauwenberghs proposes a stochastic version of this kind of learning rule (Cauwenberghs, 1993). In his algorithm, a single weight is selected randomly to add a perturbation. He analyzes the convergence of his learning rule.

On the other hand, the learning rule via a sinusoidal perturbation signal was proposed (Matsumoto & Koga, 1990). This learning rule uses multifrequency sinusoidal perturbations simultaneously. Therefore, updating of each weight is carried out simultaneously. However, this multifrequency oscillation learning method needs many detection units. Moreover, we must take cross talk into account, because the method utilizes multifrequency.

In this paper, from another point of view, we propose a learning rule via a simultaneous perturbation. Instead of the simple perturbation or the sinusoid perturbation, we introduce a simultaneous perturbation. As a result, the described learning rule requires only twice forward operations per one modification of all weights, no matter how large the neural network is. Moreover, the configuration of our rule becomes simpler. The learning rule via the simultaneous perturbation is suitable to hardware implementation because there is no need to carry out so-called backward calculation of the back-propagation method and it makes the best use of a feature of the parallel operation of neural networks. The usefulness of this kind of learning rules in neurocontrol problem has been examined (Maeda & Kanata, 1993).

The basic idea of the simultaneous perturbation method was proposed by Spall as an extension of the Kiefer–Wolfowitz stochastic approximation method (Spall, 1992). He proved that his algorithm converges to a minimum of a regression function with probability 1, under certain conditions. To guarantee theoretical convergence, his algorithm requires strict conditions on the perturbation, a gain coefficient, and a shape of the regression function. However, our emphasis is on practical usefulness as a learning rule of neural networks: practical convergence and/or feasibility for hardware implementation. From these points, the algorithm described here is simplified, compared with Spall's idea. Alspector et al. (1993) propose a parallel gradient descent method that is identical to ours. They describe superiority of this kind of learning rule for hardware implementation. Moreover, Fujita (1992) proposes trial-and-error correlation learning of neural networks. His learning rules include these learning rules in a broad sense.

In this paper, we show some computer simulations of the proposed learning rule and a comparison between this learning rule, back-propagation method, and a learning rule using the simple perturbation. In addition, we fabricated an analog neural network circuit using the learning rule. We describe details of the circuit and show some results by the circuit. This reveals a proof of feasibility of the learning rule by hardware.

2. LEARNING RULE VIA SIMULTANEOUS PERTURBATION

We use the following nomenclature in this paper.

$w_i^t (i = 1,2, \ldots, n)$: A weight at tth revision.

$\boldsymbol{w}^t = (w_1^t, w_2^t, \ldots, w_n^t)^T$: A weight vector that consists of all weights at the tth revision. $\boldsymbol{w}^t$ contains thresholds as weights with input value 1. T denotes transpose.

$\boldsymbol{w}_i^t = (w_1^t, \ldots, w_i^t + c, \ldots, w_n^t)^T$: A weight vector at tth revision with a perturbation $c (c \neq 0)$ in the ith weight.

$\boldsymbol{s}^t = (s_1^t, \ldots, s_n^t)^T$: A sign vector. $s_i^t = \pm 1$. The sign of s_i^t is randomly determined. Moreover, the sign of s_i^t is independent with the sign of the other s_j^t; that is, $\mathrm{E}(s_i^t) = 0$, $\mathrm{E}(s_i^t s_j^t) = 0$ $(i = j)$. E denotes the expectation.

o_{pj}^k: An output of the jth cell in the kth layer for the pth pattern. A superscript denotes the layer number. Especially, *out* denotes the output layer. A subscript represents the cell number in the layer.

d_{pj}: A teaching signal (i.e., desired output) of the jth cell in the output layer for the pth pattern.

We define an error function for a certain pattern p as

$$J_p(\boldsymbol{w}^t) = \frac{1}{2} \sum_j (d_{pj} - o_{pj}^{\mathrm{out}})^2. \qquad (1)$$

[1] n denotes the total number of weights.

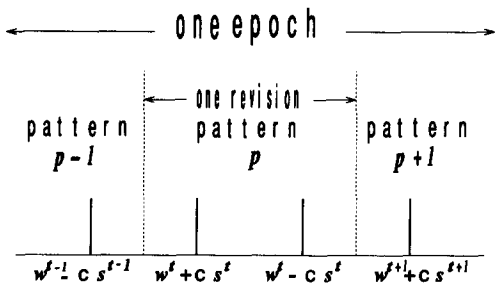

FIGURE 1. Revision. One epoch means a revision of the weight vector for all patterns.

Our problem is to find a value of the weight vector that minimizes this error function for all pattern. The gradient method, including the Newton method, is the most fundamental approach for this problem. Basically, this method needs the first derivative of the error function. Therefore, we must obtain $\partial J_p(\boldsymbol{w}^t)/\partial w_i$ for all $i(=1,\ldots,n)$. The back-propagation method performs this calculation using so-called backward error propagation.

On the other hand, we can estimate the first-differential coefficient of the error function using a difference approximation:

$$\frac{\partial J_p(\boldsymbol{w}^t)}{\partial w_i} \simeq \frac{J_p(\boldsymbol{w}_i^t) - J_p(\boldsymbol{w}^t)}{c}. \tag{2}$$

On the basis of this estimated first-differential coefficient, we can modify the weight vector as with the back-propagation method.

However, to modify all the weights in the neural network, we need $(J_p(\boldsymbol{w}_i^t) - J_p(\boldsymbol{w}^t))/c$ for all $i(=1, \ldots, n)$. This means that we need n times forward operations of the neural network. This causes a decline of an operating speed of the neural network.

In this paper, we introduce a simultaneous perturbation. In eqn (2), we added the perturbation to each weight one by one. On the other hand, we add the perturbation to all weights simultaneously. However, the sign of the perturbation is randomly determined as described by the definition of the sign vector $\boldsymbol{s}^t$ in the nomenclature. That is, by using the constant coefficient c and the sign vector $\boldsymbol{s}^t$, we consider the following quantity:

$$\Delta w_i^t = \frac{J_p(\boldsymbol{w}^t + c\boldsymbol{s}^t) - J_p(\boldsymbol{w}^t - c\boldsymbol{s}^t)}{2c} s_i^t. \tag{3}$$

We expand the right-hand side of eqn (3) at the point $\boldsymbol{w}^t$. Then, using Taylor expansion, there exist points $\boldsymbol{w}_{s1}$ and $\boldsymbol{w}_{s2}$ such that

$$\Delta w_i^t = s_i^t \boldsymbol{s}^{tT} \frac{\partial J_p(\boldsymbol{w}^t)}{\partial \boldsymbol{w}} + \frac{cs_i^t}{4} \boldsymbol{s}^{tT} \left(\frac{\partial^2 J_p(\boldsymbol{w}_{s1})}{\partial \boldsymbol{w}^2} \right) \boldsymbol{s}^t - \frac{cs_i^t}{4} \boldsymbol{s}^{tT} \left(\frac{\partial^2 J_p(\boldsymbol{w}_{s2})}{\partial \boldsymbol{w}^2} \right) \boldsymbol{s}^t. \tag{4}$$

We take an expectation of eqn (4). From the conditions of the sign vector $\boldsymbol{s}^t$, we have

$$\mathrm{E}(\Delta w_i^t) = \frac{\partial J_p(\boldsymbol{w}^t)}{\partial w_i}. \tag{5}$$

That is, Δw_i^t approximates $\partial J(\boldsymbol{w}^t)/\partial w_i$ in the sense of the expectation. We can rewrite eqn (5) as

$$\Delta w_i^t = \frac{\partial J_p(\boldsymbol{w}^t)}{\partial w_i} + \xi_i^t \tag{6}$$

where ξ_i^t is a stochastic variable with zero mean. From eqn (6), we can find the learning rule a kind of a stochastic gradient method.

As a result, we present the following learning rule

$$\boldsymbol{w}^{t+1} := \boldsymbol{w}^t - \alpha \Delta \boldsymbol{w}^t \tag{7}$$

where the ith element of $\Delta \boldsymbol{w}^t$ is defined in eqn (3). We repeat this revision for the pattern number p.

$s_i^t(J_p(\boldsymbol{w}^t + c\boldsymbol{s}^t) - J_p(\boldsymbol{w}^t - c\boldsymbol{s}^t))/(2c)$ is an estimated value of the first-differential coefficient at the tth revision. On the other hand, the coefficient α is an adequate positive number. This coefficient adjusts the magnitude of the modification.

This learning rule carries on the modification of each weight vector at every presentation of patterns. In other words, revision number is renewed; t changes to $t + 1$, when a new pattern is presented; p changes to $p + 1$ (see Figures 1 and 2). Basically, our algorithm updates all weights after each pattern is presented. Related to the work by Vogl et al. (1988), we can modify this learning rule to update the weights after all patterns have been presented. However, this modification needs many more memories. In this paper, we adopt the basic scheme. In this algorithm, there is no need to calculate the first-differential coefficient of the error function analytically. Moreover, this algorithm needs only twice feedforward operations of the neural network. It is relatively easy to realize the feedforward circuit electronically. Thus, we can easily obtain an estimated value

for p:=1 **to** p_{max} **do** (* p_{max} is a total number of patterns *)
begin
- Add the perturbation $c\boldsymbol{s}^t$ to the weight vector $\boldsymbol{w}^t$.
- Obtain a value of the error function $J_p(\boldsymbol{w}^t + c\boldsymbol{s}^t)$.
- Subtract the perturbation $c\boldsymbol{s}^t$ from the weight vector $\boldsymbol{w}^t$.
- Obtain a value of the error function $J_p(\boldsymbol{w}^t - c\boldsymbol{s}^t)$.
- Calculate a difference between these values. (* $J_p(\boldsymbol{w}^t + c\boldsymbol{s}^t) - J_p(\boldsymbol{w}^t - c\boldsymbol{s}^t)$ *)

for i:=1 **to** n **do**
begin
- Multiply this by $\alpha s_i^t/c$.

end; (* We obtained the modifying quantities for all weights. *)
- Update the weight vector. (* $\boldsymbol{w}^{t+1} := \boldsymbol{w}^t - \alpha\Delta\boldsymbol{w}^t$ *)
- Renew the iteration. (* $t := t+1$ *)

end.

FIGURE 2. Procedure of the learning rule. This shows the procedure in each epoch. We need the values of $J(\boldsymbol{w}^t + c\boldsymbol{s}^t)$ and $J(\boldsymbol{w}^t - c\boldsymbol{s}^t)$. Therefore, we require only twice forward operations of the network in each modification of the weights.

TABLE 1
Simulation Results for the Exclusive-OR Problem

	Learning Rule via the Simultaneous Perturbation		Back Propagation		Learning Rule via the Difference Approximation	
Coefficients	$\alpha = 0.8, c = 0.2$	$\alpha = 0.4, c = 0.1$	$\alpha = 0.8$	$\alpha = 0.4$	$\alpha = 0.8, c = 0.1$	$\alpha = 0.4, c = 0.1$
Convergence rate (%)	79.7	82.4	82.0	57.3	42.9	18.9
Average number of epoch for convergence	13591	16567	7809	15868	22499	25235

Total trial number is 2000 times. If the total error function $\Sigma_p J_p(\cdot) < 0.001$ or epoch was greater than 100,000, we stopped the trial.

of the first-differential coefficient using an electronic circuit. Therefore, we expect the high-speed operation and easy configuration.

Instead of the constant perturbation c, we can apply random numbers (Maeda & Kanata, 1993). In this modified version of the learning rule described here, a random number sequence in an interval $[-c\ c]$ is used as the perturbation. However, in case of a hardware implementation, we need a random numbers generator and extra analog memories corresponding to all weights. For a simplicity of the circuit configuration, we used a constant as the perturbation.

3. SIMULATION RESULTS

We emulate the proposed learning rule via a digital computer and compare results with those of the usual back propagation and the simple perturbation learning rule. The exclusive-OR problem, the TCLX problem, and 26 alphabetical characters recognition problem as a simple pattern recognition problem are considered. Input–output characteristic of each neuron is the sigmoid function $f(x) = 1/(1 + e^{-x})$. Initial values of all weights were generated randomly on $[0.1\ -\ 0.1]$. A three-layered feedforward neural network is used.

First, we handle the exclusive-OR problem. Numbers of neurons in each layer are 2, 2, and 1. Table 1 shows the convergence rate to a global minimum and the average convergence epoch by using the learning rule (7) and (3), the back-propagation method, and the learning rule via the simple perturbation. One epoch means modification of all weights for all patterns. In this table, the average convergence epochs by the learning rule described here are 13591 and 16567 with $\alpha = 0.8$, $c = 0.2$ and $\alpha = 0.4$, $c = 0.1$, respectively. Figure 3 shows a typical simulation result with $\alpha = 0.4$, $c = 0.1$. In the learning rule using the simple perturbation, the right-hand side of eqn (2) is used as an ith component of Δw^i.

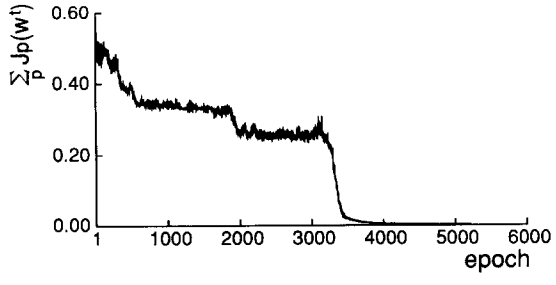

FIGURE 3. A simulation result for the exclusive-OR problem with $c = 0.1$, $\alpha = 0.4$.

The back-propagation method with $\alpha = 0.8$ is faster than the other rules. On the other hand, for smaller learning coefficient $\alpha = 0.4$, the learning rule by the simultaneous perturbation and the back-propagation method need much the same number of epochs.

As concerned with the convergence rate to a global minimum, the learning rule by the simple perturbation has the worst results. Relatively, the learning rule by the simultaneous perturbation had good results. It seems possible to improve the convergence rate by adjusting the magnitude of the perturbation.

Next, we consider the TCLX problem. Numbers of neurons in each layer are 9, 4, and 2 for the problem. Nine input signals and the combination of the two cells in the output layer represent T, C, L, and X (see Table

TABLE 2
Input Signals and Output Signals. Nine input signals show T, C, L, and X characters. The combination of the two output signals represent T, C, L, and X

		CHARACTERS			
	1 2 3 / 4 5 6 / 7 8 9	T	C	L	X
INPUT SIGNALS	i_1	1	1	1	1
	i_2	1	1	0	0
	i_3	1	1	0	1
	i_4	0	1	1	0
	i_5	1	0	0	1
	i_6	0	0	0	0
	i_7	0	1	1	1
	i_8	1	1	1	0
	i_9	0	1	1	1
TEACHING SIGNALS	d_1	0	0	1	1
	d_2	0	1	0	1

TABLE 3
Simulation Results for the TCLX Problem

	Learning Rule via the Simultaneous Perturbation		Back Propagation		Learning Rule via the Difference Approximation	
Coefficients	$\alpha = 0.8, c = 0.2$	$\alpha = 0.4, c = 0.1$	$\alpha = 0.8$	$\alpha = 0.4$	$\alpha = 0.8, c = 0.1$	$\alpha = 0.4, c = 0.1$
Convergence rate (%)	96.4	100	100	100	100	100
Average number of epoch for convergence	3678	6077	4444	8076	2224	4279

Total trial number is 2000 times. If the total error function $\Sigma_p J_p(\cdot) < 0.001$ or epoch was greater than 100,000, we stopped the trial.

2). We obtained Table 3. Figure 4 shows a simulation result with $\alpha = 0.4$, $c = 0.1$.

For this example, we could not find the remarkable difference between the back-propagation method and the learning rule by the simultaneous perturbation. However, the results by the simple perturbation were good. Totally, the learning rule described here has an equivalent capability to the back-propagation method.

Moreover, to confirm a usefulness of the learning rule using the simultaneous perturbation, we apply it to a larger neural network. We examine an alphabetical characters recognition problem. Numbers of neurons in each layer are 35 (5×7), 26, and 26. The 26 output neurons correspond to each alphabet letter. Figure 5 shows a result of a learning process with $\alpha = 0.005$, $c = 0.05$. Average convergence epoch was 34,570 for 50 trials. Average convergence rate was 36.0%. In this simulation, if epoch was greater than 50,000 or the total error function was less than 0.1, we stopped the trial.

We could obtain a neural network learning the 26 characters by using our learning rule. The network has over 1500 connections (i.e., weights). Even in this case, we require only twice forward operations of the network to obtain the modifying quantities corresponding to all weights.

4. A FABRICATION OF A NEURAL NETWORK CIRCUIT

We make an electronic neural network circuit using the algorithm (7) and (3) in trial. Figure 6 shows a picture of the fabricated neural network circuit system. Figure 7 shows a picture of a board for a weight part.

Firstly, we must decide by what we replace the input, output, and weights. In our fabrication, we replace the input, output, and the weights by voltage. The configuration of our circuit is shown in Figure 8. The neural network circuit mainly consists of two units: a neuron unit and a learning unit.

The neuron unit contains three parts: a weight part, a summation part, and a function part (see Figure 9). The weight part holds the weight value and the sign of the perturbation. Moreover, a multiplication of the weight value, which is stored in the sample hold circuit, and the input, which is given by the previous layer, is carried out. This part also contains a mechanism to renew the weight value. The summation part sums up all outputs of the weight part. The function part realizes the sigmoid function.

The learning unit gives the approximated value of the first-differential coefficient and multiplies the estimated value by $\alpha/(2c)$. The result is delivered to all weight parts in the neuron unit.

Of course, except these two units, we need a control unit that controls all timing of the teaching signals, input signals, and sample hold circuits. These signals are generated by a personal computer.

An operation of the circuit is shown in Figure 10. For a certain pattern p, the circuit operates as shown in Figure 10. The circuit repeats this procedure for each pattern.

4.1. Neuron Unit

The neuron unit embodies the forward operation of the neural network totally and renews the weight value individually. The neuron unit is described in Figure 9.

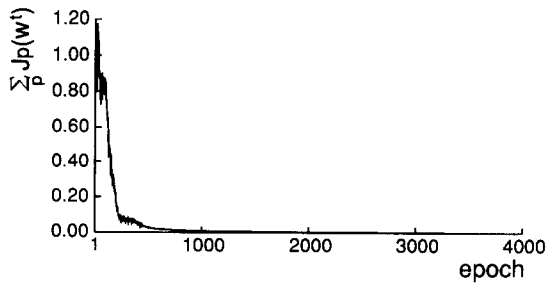

FIGURE 4. A simulation result for the TCLX problem with $c = 0.1$, $\alpha = 0.4$.

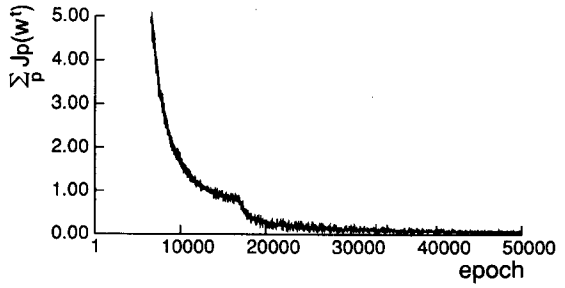

FIGURE 5. A simulation result for the 26 alphabetical characters' recognition problem with $c = 0.05$, $\alpha = 0.005$.

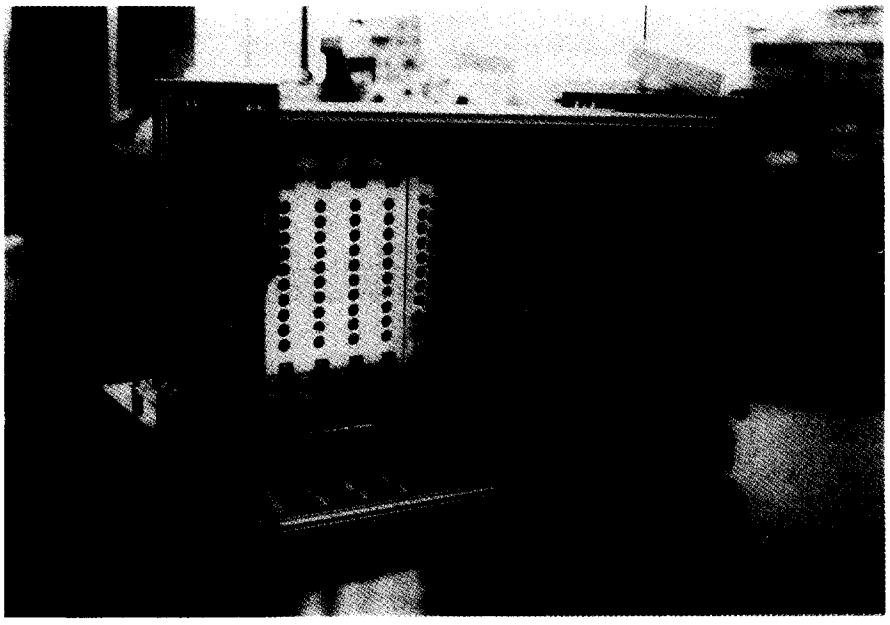

FIGURE 6. A picture of the fabricated neural network system.

The weight part memorizes the value of each weight and the sign of the perturbation in sample hold element and D-FF, respectively.

In a forward operation mode, the multiplier in this part multiplies an input that is from the previous layer (or an input of the neural network) by the corresponding weight value. This value is sent to the summation part. The summation part is composed of the usual operational amplifier. The part sums up all values. The sigmoid function is realized by the saturation property of diodes. The output of this part is connected to the other neuron units or the output of the neural network.

In a learning mode, all weight parts in the unit update the weight values in parallel by using the quantity delivered from the learning unit and the sign of the perturbation held in each D-FF. Therefore, concurrent modifications of all weights are possible.

4.2. Learning Unit

The learning unit generates the quantity $\alpha(J_p(\boldsymbol{w}^t + c\boldsymbol{s}^t) - J_p(\boldsymbol{w}^t - c\boldsymbol{s}^t))/(2c)$ and delivers the quantity to the weight part in the neuron unit.

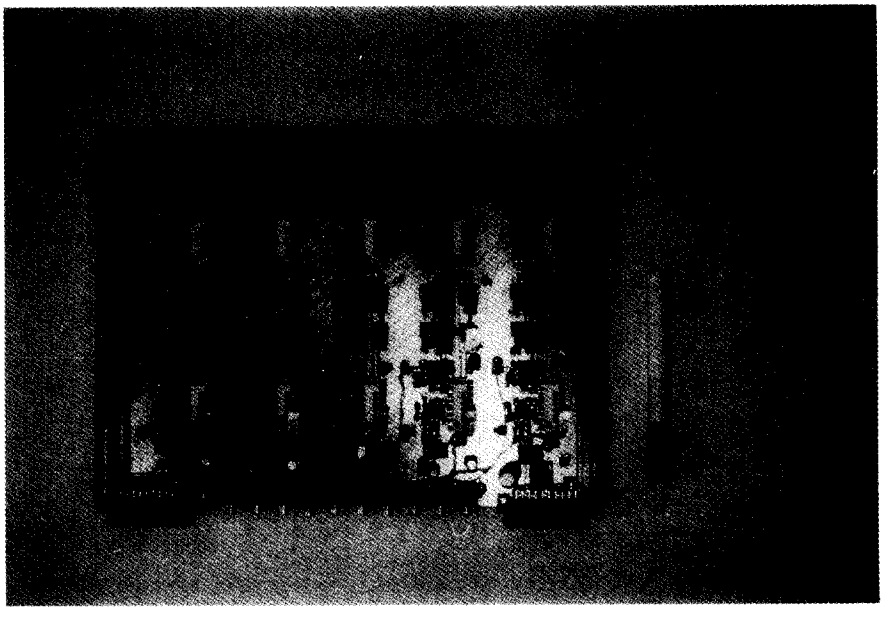

FIGURE 7. A picture of a board for weight part. In this 15 × 20 cm board, five weights are implemented.

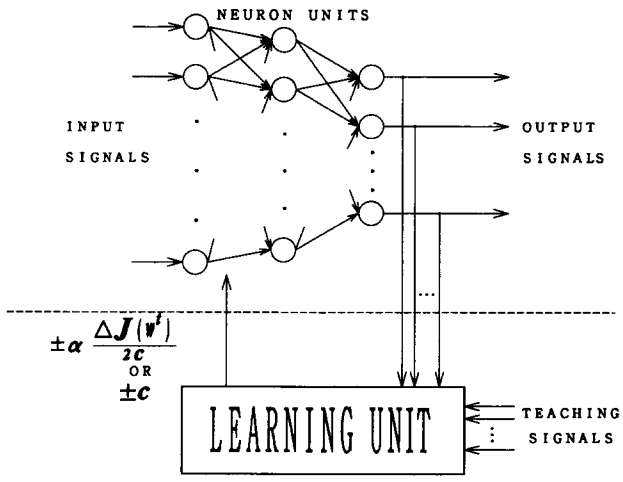

FIGURE 8. The configuration of the analog feedforward neural network circuit. The circuit consists of two units: a neuron unit and a learning unit, where $\Delta J(w^t) = J_p(w^t + cs^t) - J_p(w^t - cs^t)$.

In usual back-propagation method, the learning of neural networks is derived from so-called backward operations of the neural network. However, in our algorithm, only the forward operations of the neural network give the quantity that is used to modify the weights. Therefore, our constitution is relatively simple, compared with a straight implementation of the ordinary back propagation. Figure 11 shows the configuration of the learning unit.

Two forward operations for the weight $(w^t + cs^t)$ and $(w^t - cs^t)$ give the corresponding values $J_p(w^t + cs^t)$ and $J_p(w^t - cs^t)$ of the error function. To obtain the quantity of eqn (3), $J_p(w^t + cs^t)$ and $J_p(w^t - cs^t)$ are stored in the sample hold element of this unit, temporarily. After a calculation of a difference between

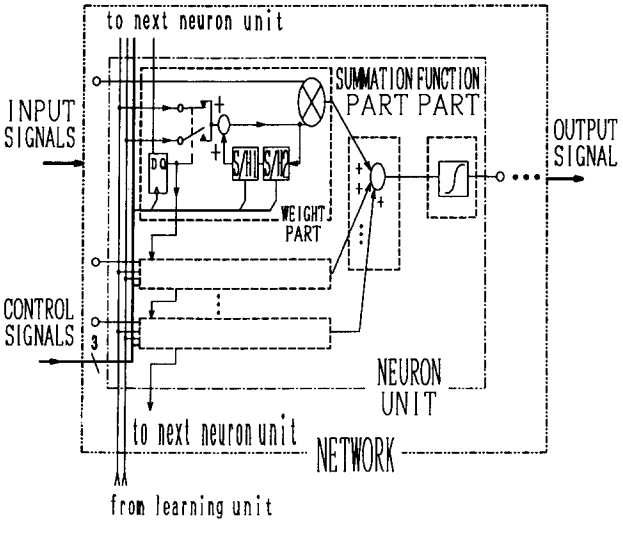

FIGURE 9. The configuration of the neuron unit. The neuron unit consists of three parts: a weight part, a summation part, and a function part.

- A sign is generated by a random digits sequence in the controlling computer. The sign is shifted and latched by each D-FF in Figure 9. (* The generation of the sign vector s^t. *)

for i:=1 **to** n **do** (* This loop is carried out simultaneously *)
begin
- On the basis of the sign of s^t_i that is latched in D-FF, we add the perturbation $+c$ or $-c$ to the weight w^t_i for all i simultaneously. (* $w^t+c\,s^t$ *)

end;
- Perform the forward operation. Then, we get the value $J_p(w^t+cs^t)$. This value is memorized in a sample hold element.

for i:=1 **to** n **do** (* This loop is carried out simultaneously *)
begin
- On the basis of the sign of s^t_i, we subtract the perturbation $+c$ or $-c$ from the weight w^t_i for all i simultaneously. (* w^t-cs^t *)

end;
- Perform the forward operation. Then, we get the value $J_p(w^t - cs^t)$. This value is memorized in a sample hold element.
- Calculate the difference between $J_p(w^t+cs^t)$ and $J_p(w^t-cs^t)$. (* $J_p(w^t+cs^t)-J_p(w^t-cs^t)$ *)
- Multiply $J_p(w^t+cs^t)-J_p(w^t-cs^t)$ by the voltage corresponding to $\alpha/(2c)$. (* $\alpha(J_p(w^t+cs^t)-J_p(w^t-cs^t))/(2c)$ *)

for i:=1 **to** n **do** (* This loop is carried out simultaneously *)
begin
- Multiply the value by the sign s^t_i for all i simultaneously. (* $\alpha\Delta w^t$ *)
- Subtract i-th component of $\alpha\Delta w^t$ from the previous value of the weight.

end. (* $w^{t+1}:=w^t - \alpha\Delta w^t$ *)

FIGURE 10. Operation of the neural network circuit. The circuit repeats this procedure for each pattern.

$J_p(w^t + cs^t)$ and $J_p(w^t - cs^t)$, we multiply the quantity $\alpha/(2c)$ by $J_p(w^t + cs^t) - J_p(w^t - cs^t)$. The result is sent to the weight parts to update the weight values.

4.3. Control Unit

This circuit needs control signals that manage all timing of sample hold elements and presentations of input signals and teaching signals and so on (see Figure 12). The control unit is composed of a personal computer.

5. RESULTS

5.1. The Exclusive-OR Problem

We get the neural network circuit to learn the exclusive-OR problem. Figure 13 shows the waves of the

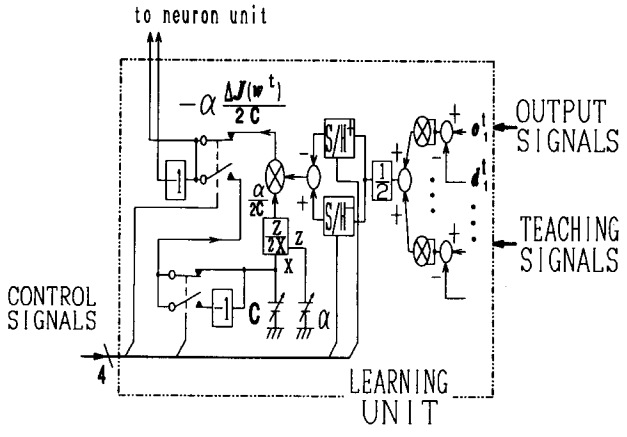

FIGURE 11. The configuration of the learning unit. The learning unit detects the squared error and generates the quantity $\alpha(J_p(w^t + cs^t) - J_p(w^t - cs^t))/(2c)$.

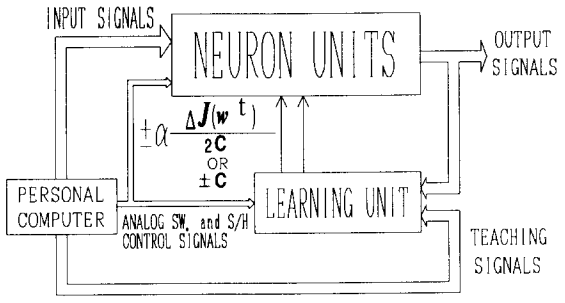

FIGURE 12. The flow of the control signals. The control unit generates timing signals of sample hold elements, input signals, teaching signals, and so on.

learning pattern (input signals and teaching signal) and the observed output of our circuit. In a period T, the control part gives the four patterns of the exclusive-OR to this neural network circuit. The good agreement between the observed output of our neural network circuit and the teaching signal shows that the circuit learned the exclusive-OR problem.

In a period $T/4$, that is, in a period presenting a certain pattern, there are nine modifications of the weight.

In these results, T is about 11.1 ms. In other words, it takes about 2.8 ms to learn one pattern in this case. In Figure 13, the operation speed is approximately 3.2 kcups. On average, we need about 2–3 min to obtain a stable result (for example, shown in Figure 13). This time corresponds to 10,000–15,000 epochs.

Figure 14 shows that the circuit is captured in so-called local minimum. With the difference between the teaching signal and the practical output, the output of this circuit is stable. Our neural network circuit learns OR, though the teaching signal is the exclusive-OR.

In such a case, we adjust the values of coefficients. Potentially, our leaning rule contains an ability to pass through the local minimum. The modifying quantity defined in eqn (3) consists of the first-differential coefficient and an another error as described in eqn (6).

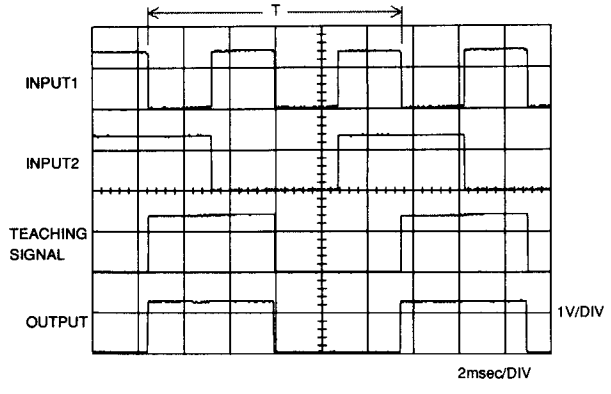

FIGURE 13. A result for the exclusive-OR problem. The agreement between the teaching signal and the observed output shows that the circuit learns the exclusive-OR problem.

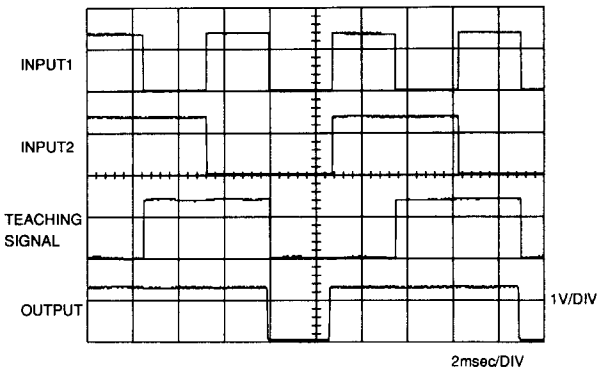

FIGURE 14. A local minimum. The circuit is captured in a local minimum. The circuit learns the OR relation.

If the weight w_i is located in the neighborhood of a certain minimum point, from eqn (4), the perturbation c prescribes the magnitude of the error. Thus, the larger c is, the larger the error is, and vice versa if c is smaller. From this point of view, the learning rule has a property like the simulated annealing. By adding an adequate mechanism, we can obtain the neural network circuit that passes through the local minimums in some measure. However, we need detailed discussions, analysis and experiences for this point.

5.2. The TCLX Problem

Secondly, we consider the TCLX problem. The number of cell in each layer is 9-4-2. The number of weights including thresholds is 50.

Figure 15 shows the teaching signals and the observed outputs for this problem. This figure shows that the neural network circuit learns the TCLX patterns. Also in this problem, the modifications of all weights are performed for a period $T/4$. That is, 50 weights are updated for 8.4 ms. In this case, we need 2–3 min to obtain a result. This corresponds to 3500–5500 epochs. The operation speed in this figure is approximately 6.0

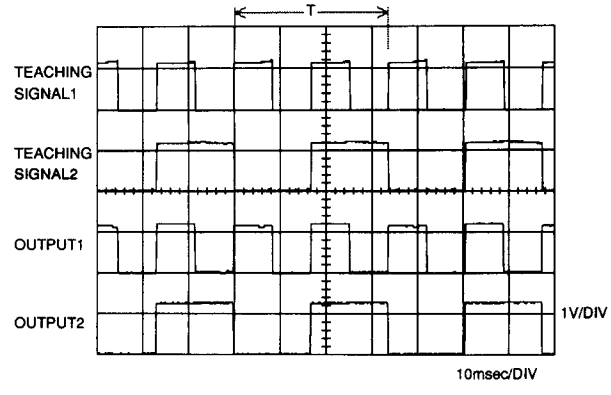

FIGURE 15. A result for the TCLX problem. The circuit learns the TCLX problem.

kcups. This speed depends on a clock frequency of the circuit.

At this stage, we use individual parts, that is, operational amplifiers, sample hold ICs, switching ICs, and so on. Basically, we can replace these devices by LSI. Such an integration will contribute to the high-speed operation and stability of our circuit.

6. CONCLUSION

In this paper, we described the learning rule of neural networks using the simultaneous perturbation. We showed some computer simulations of the rule and a comparison between this method and the other methods. This learning rule needs only the forward operations of neural networks. Therefore, this learning rule is superior to the conventional back-propagation method for large networks. Moreover, it is relatively easy to realize it electrically. We fabricated the analog feedforward neural network circuit with learning ability by using the proposed learning rule. The circuit learned the exclusive-OR problem and the TCLX problem. We stated the details and the operation results of the analog neural network circuit.

This learning rule is useful not only for multilayer feedforward neural networks but also for recurrent neural networks. However, a more detailed discussion and experiences are needed.

REFERENCES

Alspector, J., Meir, R., Yuhas, B., Jayakumar, A., & Lippe, D. (1993). A parallel gradient descent method for learning in analog VLSI neural networks. In S. J. Hanson, J. D. Cowan, & C. Lee (eds.), *Advances in Neural Information Processing Systems 5* (pp. 836–844) San Mateo, CA: Morgan Kaufmann Publishers, Inc.

Cauwenberghs, G. (1993). A fast stochastic error-descent algorithm for supervised learning and optimization. In S. J. Hanson, J. D. Cowan, & C. Lee (Eds.), *Advances in Neural Information Processing Systems 5* (pp. 244–251) San Mateo, CA: Morgan Kaufmann Publishers, Inc.

De Gloria, A. (1989). The VLSI technology and the design of neural networks. In E. R. Caianiello (Ed.), *Parallel Architectures and Neural Networks.* Teaneck, NJ: World Scientific.

Eberhardt, S. P., Tawel, R., Brown, T. X., Daud, T., & Thakoor, A. P. (1992). Analog VLSI neural networks: Implementation issues and examples in optimization and supervised learning. *IEEE Transactions on Industrial Electronics,* **39,** 552–564.

Fujita, O. (1993). Trial-and-error correlation learning. *IEEE transactions on Neural Networks,* **4,** 720–722.

Jabri, M., & Flower, B. (1992). Weight perturbation: An optimal architecture and learning technique for analog VLSI feedforward and recurrent multilayer networks. *IEEE Transactions on Neural Networks,* **3,** 154–157.

Maeda, Y. (1992). Learning rules of neural networks for inverse systems. *Transactions of The Institute of Electronics, Information and Communication Engineers,* **J75-A,** 1364–1369 (in Japanese), English version of this paper is shown in *Electronics and Communications in Japan (1993),* **76,** 17–23.

Maeda, Y., & Kanata, Y. (1993). Learning rules for recurrent neural networks using perturbation and their application to neuro-control. *Transactions of The Institute of Electrical Engineers of Japan,* **113-C,** 402–408 (in Japanese).

Maeda, Y., Yamashita, H., & Kanata, Y. (1991). Learning rules for multilayer neural networks using a difference approximation. *Proceedings of 1991 International Joint Conference on Neural Networks Singapore,* **1,** 628–633.

Matsumoto, T., & Koga, M. (1990). Novel learning method for analogue neural networks. *Electronic Letters,* **26,** 1136–1137.

Mead, C., & Ismail, M. (Eds.) (1989). *Analog VLSI implementation of neural systems.* New York: Kluwer Academic Publishers.

Rumelhart, D. E., McClelland, J. L., & The PDP group (1986). *Parallel distributed processing.* Cambridge, MA: The MIT Press.

Spall, J. C. (1992). Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control,* **37**(9), 332–341.

Vogl, T. P., Mangis, J. K., Rigler, A. K., Zink, W. T., & Alkon, D. L. (1988). Accelerating the convergence of the back-propagation method. *Biological Cybernetics,* **59,** 257–263.