

FPGA Implementation of a Pulse Density Neural Network With Learning Ability Using Simultaneous Perturbation

Yutaka Maeda and Toshiki Tada

Abstract—Hardware realization is very important when considering wider applications of neural networks (NNs). In particular, hardware NNs with a learning ability are intriguing. In these networks, the learning scheme is of much interest, with the backpropagation method being widely used. A gradient type of learning rule is not easy to realize in an electronic system, since calculation of the gradients for all weights in the network is very difficult. More suitable is the simultaneous perturbation method, since the learning rule requires only forward operations of the network to modify weights unlike the backpropagation method. In addition, pulse density NN systems have some promising properties, as they are robust to noisy situations and can handle analog quantities based on the digital circuits. In this paper, we describe a field-programmable gate array realization of a pulse density NN using the simultaneous perturbation method as the learning scheme. We confirm the viability of the design and the operation of the actual NN system through some examples.

Index Terms—Field-programmable gate array (FPGA), learning ability, neural networks (NNs), pulse density, simultaneous perturbation.

I. INTRODUCTION

NEURAL NETWORKS (NNs) are widely used in a number of applications in which the NNs are usually implemented as a software program on an ordinary digital computer. However, software implementations cannot utilize the essential property of parallelism found in biological NNs. In this respect, implementation of NNs using hardware elements such as very large-scale integration (VLSI) is beneficial.

When considering the hardware implementation of an NN, realization of the learning mechanism as a hardware system is an important and difficult issue [1]. As we well know, the backpropagation method is commonly used. However, realization of the backpropagation method as an electronic system is very difficult, considering wiring for modifying quantities to all weights, calculation of the derivative of the sigmoid function, and so on.

Thus, it is particularly difficult to implement large-scale NNs with learning ability via the gradient method because of the complexity of the mechanism that derives the gradient. From this point of view, we must try to find a learning rule that is easy to realize.

The simultaneous perturbation method was introduced by Spall [2], [3], Alespector *et al.* [4], and Cauwenberghs [5]. Maeda also independently proposed a learning rule of NNs using simultaneous perturbation and reported a feasibility of

the learning rule [6]–[8]. At the same time, the merit of the learning rule was demonstrated in VLSI implementation of analog NNs [9], [10].

The advantage of the simultaneous perturbation optimization method is its simplicity. The method can estimate the gradient using only values of the error function. Therefore, implementation of this learning rule is relatively easy compared to that of other learning rules, because it does not have to take the error backpropagation circuit into account.

Certain pulse techniques, such as pulse width or pulse stream, have also been investigated to implement artificial NNs. For example, El-Masry *et al.* reported an efficient implementation of artificial NNs using a current-mode pulse width modulation artificial NN [11]. Moreover, Murray *et al.* proposed a VLSI NN using analog and digital techniques [12].

In particular, pulse density NNs have fascinating properties. For example, pulse systems are invulnerable to noisy conditions. Moreover, pulse density systems can handle quantized analog values based on the digital circuit [13]. Based on these features, Hikawa reported a frequency-based NN using the backpropagation [14]. In [14], the ordinary backpropagation method is applied to a pulse density NN.

However, it seems difficult to employ the backpropagation method for a pulse density system. Actually, NN system described in [14] has to complete the error propagation mechanism based on the pulse density, in which case the circuit design becomes complex compared with the simultaneous perturbation method.

Recently, field programmable gate arrays (FPGAs) have been used in many commercial fields because of their reconfiguration properties and flexibility [15]. FPGAs also seem to be promising devices for implementing NNs, in comparison with ordinary software implementations.

VHDL is a very popular hardware description language (HDL) for describing or designing digital circuits. In the fundamental design of this research, HDL is used.

Combining a pulse density system with the simultaneous perturbation method, we can easily design analog hardware NN systems with learning capability. Some of the features of a pulse density NN system using FPGA can be summarized as follows: 1) Hardware can take advantage of parallelism; 2) simultaneous perturbation learning rule is very simple; 3) analog NN system is realized based on digital circuits; 4) digital design technology used is supported by electronic design automation; and 5) pulse density NNs are not affected by noisy situations.

II. SIMULTANEOUS PERTURBATION LEARNING RULE

Details of the simultaneous perturbation method as a learning rule of NNs have been described previously [6]–[9], [13] and are reiterated in this section.

Manuscript received October 18, 2001; revised March 4, 2002 and January 3, 2003. This work was supported in part by Kansai University High Technology Research Center.

The authors are with the Department of Electrical Engineering, Faculty of Engineering, Kansai University, Osaka 564-8680, Japan.

Digital Object Identifier 10.1109/TNN.2003.811357

The simultaneous perturbation learning rule with a sign vector is described as follows:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \Delta \mathbf{w}_t \quad (1)$$

$$\Delta w_t^i = \frac{J(\mathbf{w}_t + c\mathbf{s}_t) - J(\mathbf{w}_t)}{c} s_t^i \quad (2)$$

where \mathbf{w}_t is the weight vector including the threshold of all neurons in a network at the t th iteration. c is the magnitude of the perturbation. α is a positive constant to scale a modifying quantity, called the learning coefficient. $\Delta \mathbf{w}_t$ is a modifying vector and Δw_t^i represents the i th element of the vector $\Delta \mathbf{w}_t$. \mathbf{s}_t and s_t^i denote a sign vector and its i th element that is 1 or -1 , respectively. Therefore, if the sign s_t^i is $+1$, a positive perturbation $+c$ is added to the i th weight, if it is -1 , a negative perturbation $-c$ is used. The coefficient c controls the magnitude of the perturbations. The sign of s_t^i is randomly determined with a zero mean for every iteration. Moreover, the sign of s_t^i is independent of the sign of the j th element s_t^j of the sign vector. This means that a different sign is used for a different weight. That is

$$E(s_t^i) = 0, \quad E(s_t^i s_t^j) = 0 (i \neq j) \quad (3)$$

where E denotes the expectation.

The error function $J(\mathbf{w})$ is defined by an output of the NN o and a corresponding teaching signal d as follows:

$$J(\mathbf{w}) = |o - d|. \quad (4)$$

Ordinarily, the error function is defined by a squared error. However, we use the absolute error for circuit simplification.

In this learning rule, the modifying quantities for all weights in the network are calculated using two values of the error function $J(\mathbf{w})$ and $J(\mathbf{w} + c\mathbf{s})$. Two forward operations of the network give $J(\mathbf{w} + c\mathbf{s}) - J(\mathbf{w})$. Multiplying this by s^i/c yields (2). This quantity is an estimated gradient of the error function with respect to w^i . Repeating this for $i = 1, \dots, n$, we update all the weights.

When we expand the error $J(\mathbf{w}_t + c\mathbf{s}_t)$ at the point \mathbf{w}_t , there exist \mathbf{w}_{s1} such that

$$J(\mathbf{w}_t + c\mathbf{s}_t) = J(\mathbf{w}_t) + c\mathbf{s}_t^T \frac{\partial J(\mathbf{w}_t)}{\partial \mathbf{w}} + \frac{c^2}{2} \mathbf{s}_t^T \frac{\partial^2 J(u(\mathbf{w}_{s1}))}{\partial \mathbf{w}^2} \mathbf{s}_t. \quad (5)$$

Therefore, (2) becomes

$$\Delta w_t^i = s_t^i \mathbf{s}_t^T \frac{\partial J(\mathbf{w}_t)}{\partial \mathbf{w}} + \frac{cs_t^i}{2} \mathbf{s}_t^T \frac{\partial^2 J(u(\mathbf{w}_{s1}))}{\partial \mathbf{w}^2} \mathbf{s}_t. \quad (6)$$

We take an expectation of the above quantity. From the conditions of the sign vector \mathbf{s}_t in (3), we have

$$E(\Delta w_t^i) = \frac{\partial J(\mathbf{w}_t)}{\partial w_t^i}. \quad (7)$$

That is, Δw_t^i approximates $\partial J_p(\mathbf{w}_t)/\partial w_t^i$. Since the right-hand side of (2) is an estimated value of the first-differential coefficient, the learning rule is a type of a stochastic gradient method [8], [9].

An important point is that this learning rule requires only two values of the error function. That is, it requires only two forward

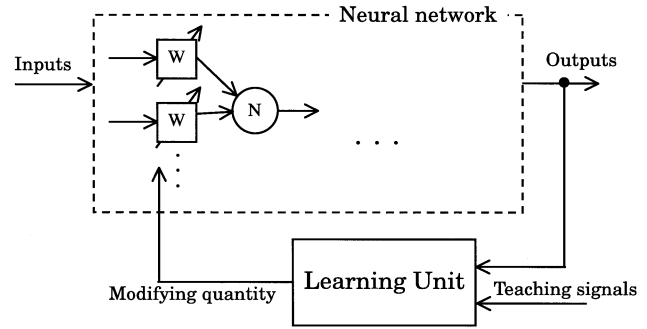


Fig. 1. Pulse density NN via simultaneous perturbation.

operations of the NN in order to obtain estimators of the first differential coefficients of the error function with respect to all weights in the network.

From the point of view of realizing the learning ability of NNs, the simplicity and ease of the simultaneous perturbation method is highly beneficial. If we use the ordinary backpropagation method, we have to include the so-called error-propagation circuits for all weights in the network. The error-propagation through the weights, the wiring for all weights in the network, and the overall circuit design becomes difficult. The simultaneous perturbation learning rule does not involve the error-propagation, and only values of the error function are required to update all the weights. Therefore, only one circuit realizing the learning mechanism is used to update all the weights, even if the number of weights is large.

Moreover, a pulse density system assists in simplifying the design. In pulse density NNs, the number of pulses denote outputs of the neurons and, hence, output of the network. Therefore, ordinary up-down counters can implement addition or subtraction. For example, counting up for the output of the network and counting down for a teaching signal yields the error of (4). Similarly, the denominator of the right-hand side of (2) can be easily realized with a counter. The main part of the learning mechanism can be implemented with ordinary logical elements used in common digital circuits.

III. BASIC ARCHITECTURE AND IMPLEMENTATION

It is crucial to implement NNs using suitable media in order to take advantage of some ideas of real biological nerve systems.

The overall configuration of our NN is shown in Fig. 1. In this figure, W and N denote a single weight unit and a single neuron unit, respectively. Basically, the system consists of three kinds of units: weight units, neuron units, and a single learning unit. The NN itself is composed of multiple weight units and multiple neuron units. Only one learning unit generates the basic quantity of modification for all weights and conveys it to all weight units.

A. Weight Unit

The weight unit is shown in Fig. 2. This unit calculates the product of the input signals and the weight value. At the same time, the unit updates the weight value.

The weight unit consists of a weight modification part and a random-number generation part. The weight modification part updates the weight value based on the quantity from the learning

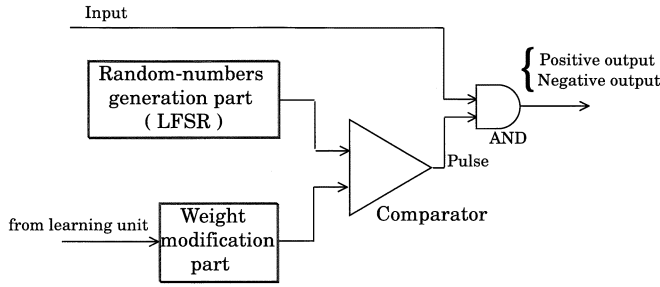


Fig. 2. Weight unit.

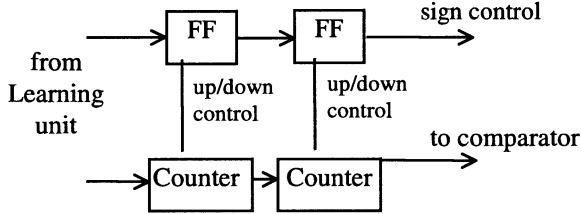


Fig. 3. Weight modification part.

unit and carries out addition or subtraction of the perturbation. At the same time, it stores the weight value. The random-number generation part generates a random number using a linear feedback shift register.

If the sign of the result of the unit is positive, the output is sent to the positive side of the neuron unit. If the sign is negative, the output is sent to the negative side.

1) *Weight Modification*: Fig. 3 depicts the weight modification part. The first counter (eight bits) and the first Flip Flop (FF) in this part (left counter and FF in Fig. 3) store an initial value of a weight and its corresponding sign, respectively. The basic modifying quantity $\alpha(J(w+cs) - J(w))/c$ in (2) is common to all weights. This quantity is sent from the learning unit, and connected to the first counter. The sign of the quantity is connected to the first FF. The sign s_i^j in (2), which is generated by the linear feedback shift register, is also connected to the FF which decides whether counting up or down should be performed. These operations modify the weights as represented in (2).

Another role of the weight modification part is to add a perturbation to the weight. This is simultaneously done for all weights in each weight modification part. The second counter and the second FF (right counter and FF in Fig. 3) are used for this purpose. That is, the perturbation c , which is constant, is added by the counter, and the sign of the perturbation is stored in the second FF.

2) *Pulse Generation*: The weight values calculated in the weight modification part must be converted into a pulse series. We use a random-number generator and a comparator for this. The linear feedback shift register is used to produce random numbers. We compare a weight value with a random value generated by the linear feedback shift register. If the weight is larger than the random number, this circuit generates a single pulse and if not, no pulse is generated. We repeat this procedure, and new random numbers are generated at each time step. Therefore, a large weight results in many pulses and a small weight results in very few pulses. In other words, the weights in our system are represented by pulse density.

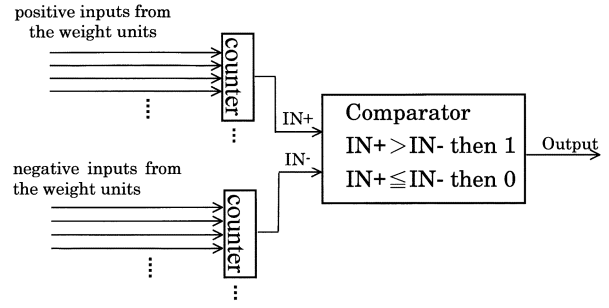


Fig. 4. Neuron unit.

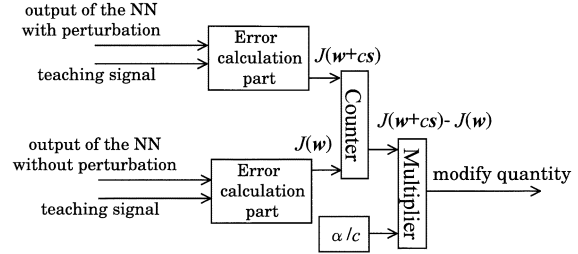


Fig. 5. Learning unit.

B. Neuron Unit

Fig. 4 shows the neuron unit which consists of counters and a comparator and calculates the weighted sum of inputs. The counters sum the number of pulses given by the weight units as shown in Fig. 4. The first counter (upper counter in Fig. 4) counts the number of positive inputs, and the second counter (lower counter in Fig. 4) counts the number of negative inputs. If the number of positive pulses is larger than the number of negative pulses, the neuron unit generates a single pulse. The input–output behavior of our neuron units is characterized by a piecewise-linear function determined by the saturation of pulse density. That is, even if a weighted sum for a neuron is extremely large, the maximum number of pulses per unit time is limited. No pulse indicates the weighted sum of a neuron is less than the lowest limit of the output. Otherwise, the number of the output pulses is equal to the weighted sum of inputs. That is, the amplification factor of the linear function is assumed to be unity. Thus, instead of the sigmoid function, the system uses a linear function with a restriction applied. A similar idea for pulse density neurons is discussed in [14].

C. Learning Unit

The learning unit achieves the so-called learning process using simultaneous perturbation and sends the basic modifying quantity $\alpha(J(w+cs) - J(w))/c$ to the weight units, which is common to all weights. The block diagram is shown in Fig. 5. One of the features of this learning rule is that it requires only forward operations of the NN.

There is a counter in each error calculation part. Since the error function used here is defined by the absolute difference as in (4), using the counter, this part gives the difference in the number of pulses between the output of the NN and the corresponding teaching signal; counting up for the output pulses and counting down for the teaching pulses provide the error. The

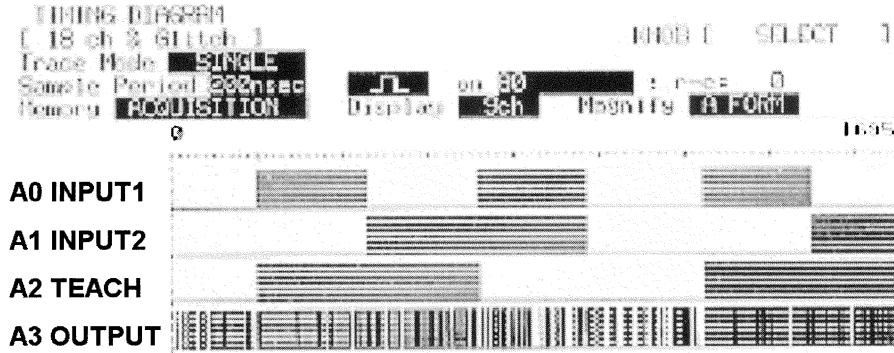


Fig. 6. Result before learning.

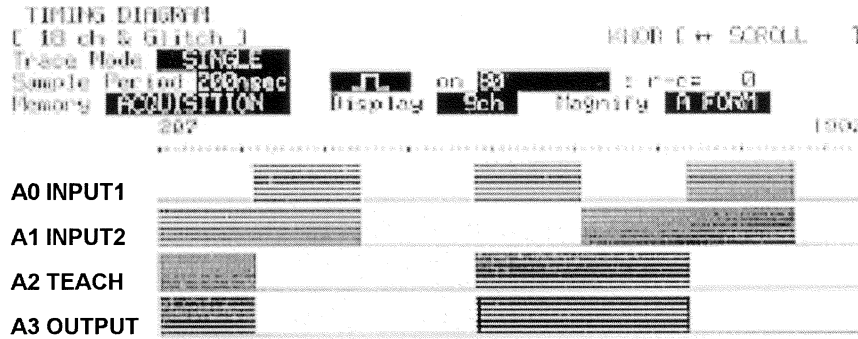


Fig. 7. Result after learning for the exclusive OR problem.

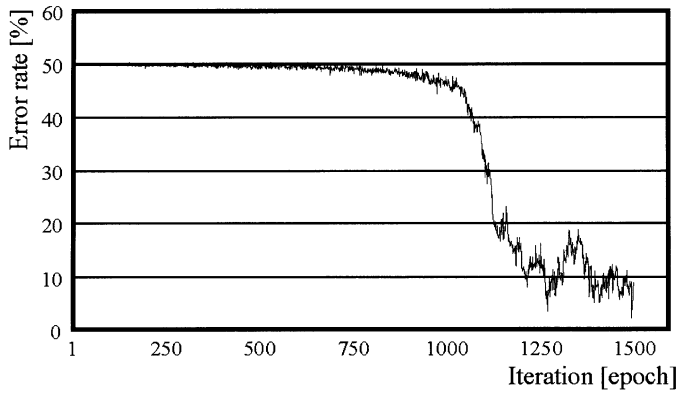


Fig. 8. Error rate for the exclusive OR problem.

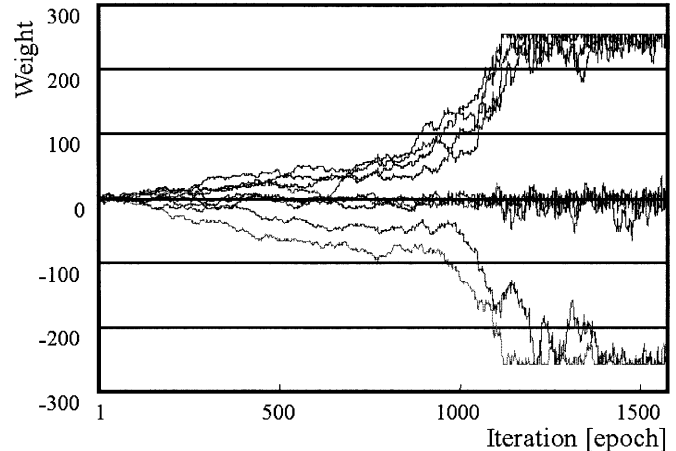


Fig. 9. Weights of the network for the exclusive OR problem.

two error calculation parts yield two values of the error function; $J(\mathbf{w} + c\mathbf{s})$ and $J(\mathbf{w})$. The next counter in Fig. 5 gives the difference between these two values: $J(\mathbf{w} + c\mathbf{s}) - J(\mathbf{w})$. The learning coefficient α and the perturbation c are constant. Multiplying by α/c , the unit computes the basic modifying quantity, which is sent to all weight units.

IV. EXAMPLES

In this chapter, we describe some concrete examples of exclusive OR learning, function learning and the TCLX problem. The networks are designed to solve these three problems.

The design results were checked and configured using the logical synthesis software MAX PLUS. ALTERA, FLEX EPF10K250AGC599-3 with a 20-MHz clock speed is used for these examples.

A. Exclusive OR

First, a three-layered NN is configured to learn exclusive OR. The numbers of neurons in each layer are two, two, and one. About 60 000 gates out of 250 000 gates in this FPGA are used.

The inputs, teaching signals, and the output of the NN are shown in Figs. 6 and 7. Logical representations of zero and one are shown in pulse density form. 255 ($=2^8 - 1$) pulses per unit time denotes one, and no pulse in a unit time means zero. Initial weights of the network are determined randomly in the interval $[-255 + 255]$. The magnitude of the perturbation c is 5, and the learning coefficient α is 1.25 in (1) and (2).

A black region in the figures represents an area with many pulses. Fig. 6 shows the output of the NN before learning. The

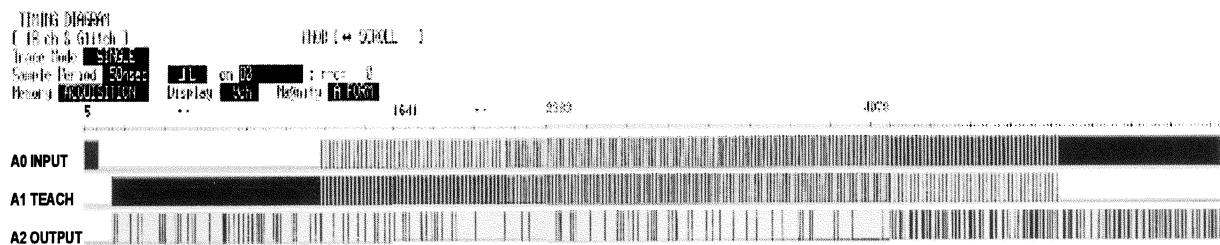


Fig. 10. Result before learning for the function learning.

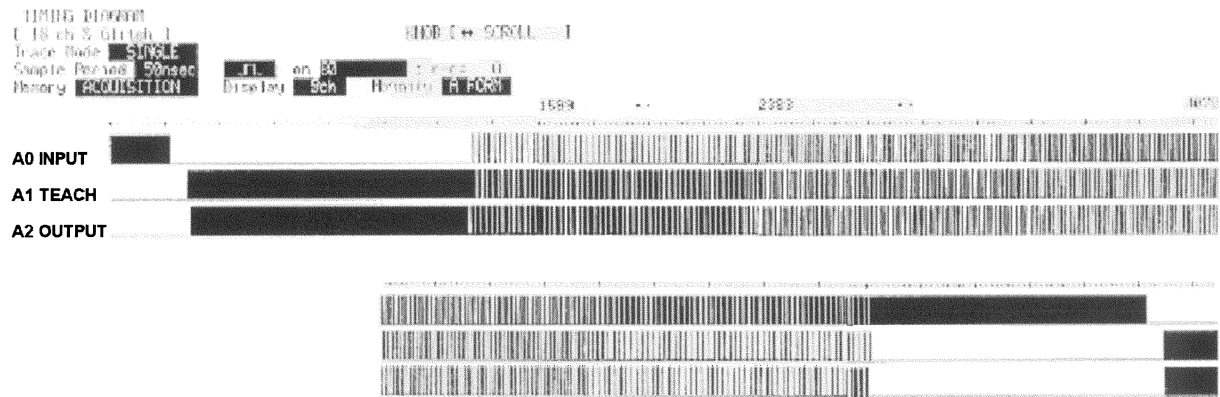


Fig. 11. Result after learning for the function learning.

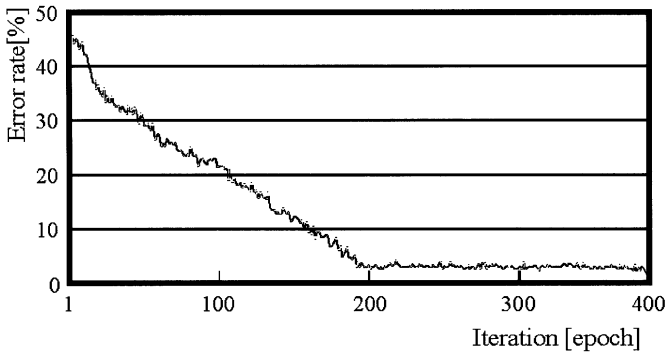


Fig. 12. Error rate for the function learning.

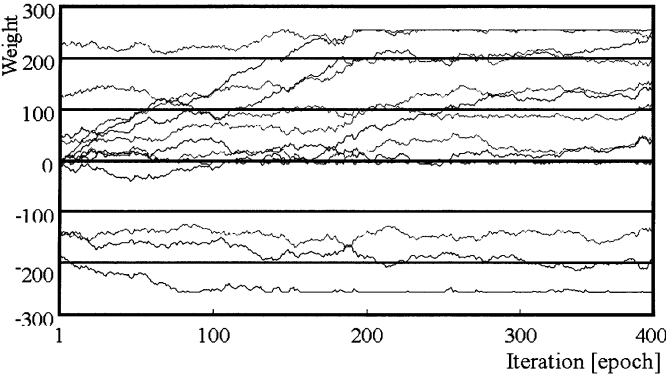


Fig. 13. Weights of the network for the function learning.

four rows show the first input, the second input, the teaching signal, and the output of the network, respectively. The behavior of the output pulses is quite different from that of the teaching

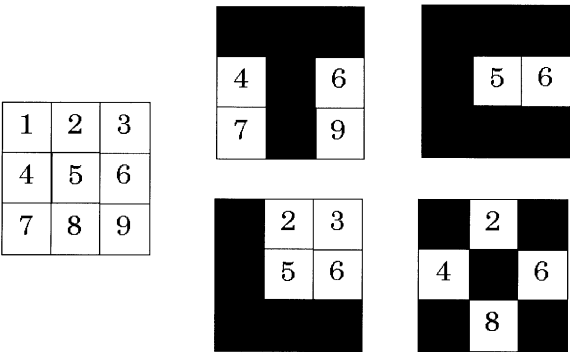


Fig. 14. Characters T, C, L, and X.

signal. After sufficient learning, we notice close agreement between the output and the teaching signals in Fig. 7. The NN learns the logical relation of exclusive OR in the form of pulse density.

The change of error rate is shown in Fig. 8. The error rate is defined by the difference between the actual number and the desired number of pulses. The perturbation c and the learning coefficient α are 5 and 2.5, respectively. We can see that the error decreases as learning proceeds. It took 55.05 μ s for one epoch, which represents the learning of four patterns of the exclusive OR relation. The learning speed was about 172.9 kCUPS.

The change of weight values in the network is shown in Fig. 9. As the learning proceeds, all weights converge to corresponding constant values.

In our algorithm, we have to determine the learning coefficient α and the perturbation c properly. It is impossible or extremely difficult to find optimal values for α and c as these values vary widely for different problems.



Fig. 15. Result before learning for the TCLX problem.

In our experiments, these values are determined empirically. For example, different values for Figs. 7 and 8 were used to obtain typical results, and from preliminary experiments, we could relatively easily find rough values of the learning coefficient and the perturbation.

Generally, a small perturbation gives a good approximation of the derivative of the error function. However, the hardware system has a lower limit for the perturbation, and too small a perturbation makes the system unstable. That is, the modification quantity $\alpha(J(w+cs) - J(w))/c$ becomes too large because of the small perturbation c .

The larger the learning coefficient α is, the faster the convergence speed. However, too large a learning coefficient leads to a poor modification quantity because of an estimation error of the derivative. From this point of view, a small coefficient is more reliable, at the expense of a slower convergence to an optimal state.

There is a tradeoff in finding suitable values for the learning coefficient and the perturbation. In these experiments, we use heuristics to determine suitable values empirically.

B. Function Learning

Next, we consider the problem of learning the function $y = 1 - x$. The size of the network is 1-4-1, using about 75 000 gates in the FPGA.

For this problem we have to handle an analog quantity, which is represented by the pulse density. $255(=2^8 - 1)$ pulses per unit time denotes one, and no pulse per unit time denotes zero. For example, 26 pulses in a unit time represents $26/255$, that is, about 0.1. The position of the pulses is not important; only the number of pulses in a unit time interval represents the analog quantity.

Initial weights of the network are determined randomly in the interval $[-255 + 255]$, and output pulses are randomly generated. Therefore, the pulse density of the output will be quite different from the corresponding teaching signals.

The inputs, teaching signals, and the output of the NN are shown before learning in Fig. 10 and after learning in Fig. 11. For the input in Fig. 10, there is no pulse in the first interval, corresponding to analog zero. In the second time interval, the number of pulses increases, and by the final interval, the input in the figure is painted over. This means the analog value increases as the time interval moves to the right. Actually, six unit time intervals, 0, 0.2, 0.4, 0.6, 0.8, and 1 are used. That is, we select six learning points, where the corresponding ideal number of pulses are $0(=0 \times 255)$, $51(=0.2 \times 255)$, $102(=0.4 \times 255)$, $153(=0.6 \times 255)$, $204(=0.8 \times 255)$ and $255(=1 \times 255)$.

Since the function is $y = 1 - x$, corresponding teaching signals are 1, 0.8, 0.6, 0.4, 0.2, and 0, respectively. We can see full pulses in the leftmost interval of the teaching signals (the second row) in both Figs. 10 and 11. The number of pulses decreases as the corresponding teaching signal decreases. After sufficient learning (Fig. 11), the number of pulses in the output for each unit interval are close to that of the corresponding teaching signals. For this result, the magnitude of the perturbation and the learning coefficient are five and 1.25, respectively.

Next, we evaluate the results. As described above, since the position of the pulses is not important, the error rate is the difference between the number of pulses in the actual output and the corresponding desired output. We measured the average number of pulses for six learning values over 15 trials. The results were zero, 51.47, 102.47, 153.00, 203.93, and 254.67 pulses on average, which compare very closely to the ideal numbers of pulses written above. The errors are all within one pulse on average.

We measured the error from an initial state. The change of error rate is shown in Fig. 12, which indicates the error decreases as the learning proceeds. The perturbation and the learning coefficient are five and 2.5, respectively. It took $77.75 \mu s$ for one epoch, which includes calculations of modification quantities for six patterns. The learning speed was about 167.2 kCUPS.

The change of weight values in the network are shown in Fig. 13. As the learning proceeds, all weights converge to corresponding constant values.

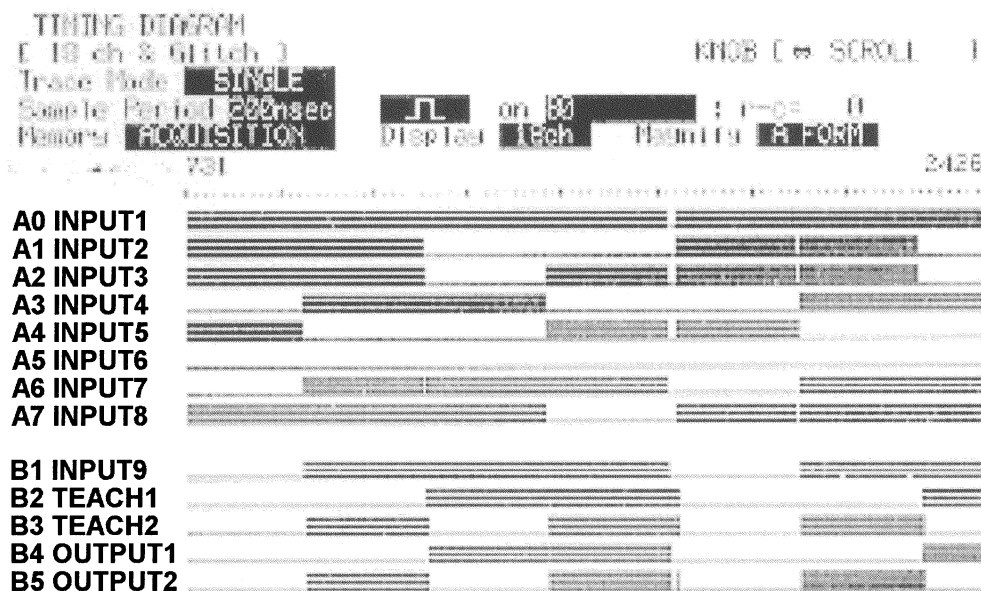


Fig. 16. Result after learning for the TCLX problem.

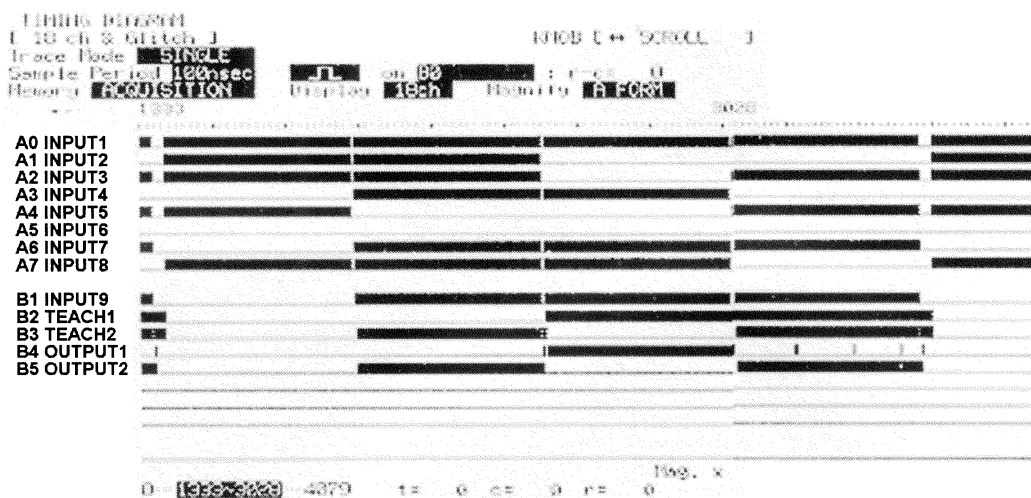


Fig. 17. Local minimum for the TCLX problem.

C. TCLX

The third example is a simple character recognition problem known as the TCLX problem. A 3×3 array of pixels is used to represent the characters T, C, L, and X (see Fig. 14). The number of neurons in each layer is 9-4-2, and the total number of weights including thresholds in the network is 50. Approximately 230 000 gates are used for the network.

The inputs, teaching signals and the output of the NN are shown in Figs. 15 and 16 with a perturbation c of 3 and a learning coefficient α of 0.375. The initial weights in the network are all zero.

A black pixel and a white pixel in Fig. 14 denote one and zero, respectively. Nine inputs correspond to the nine pixels of the pattern. For example, if the pattern is T, then 1, 1, 1, 0, 1, 0, 0, 1, and 0 are applied to the network in pulse density form (see Fig. 14). The leftmost time interval in Fig. 15 shows the inputs 1, 1, 1, 1, 0, 0, 1, 1, and 1, therefore the presented pattern is C.

A combination of two outputs represents the characters. For example, an output of zero and zero means T, of zero and one means C, and so on.

Fig. 15 shows the output of the NN before learning. Notice that the output pulses are quite different from the teaching signals. After sufficient learning, these agree closely, as shown in Fig. 16. It took $52.5 \mu\text{s}$ for one epoch, making the learning speed approximately 952.4 kCUPS.

In some cases, learning is suspended in so-called local minima, in which the NN cannot produce the desired outputs despite continued learning. Such a situation is shown in Fig. 17. For the first three patterns, both teaching signals are the same as the outputs of the NN. However, for the fourth pattern, one output is nearly zero, and the other output is one, despite the fact that the two corresponding teaching signals are one and one.

Economizing the number of required gates or the size of the circuit is possible, however we have not paid attention to this. This is a very technical aspect of digital circuit design, which is not the main theme of this paper. Compared with the ordinary backpropagation learning rule, the number of required gates in our method is small, because only a single learning circuit is commonly used to generate the modifying quantity for all weights.

V. CONCLUSION

In this paper, we designed a pulse density NN system on FPGA using the simultaneous perturbation learning rule. VHDL was used for the design of the networks, and we confirmed the system worked well for the exclusive OR, function learning, and TCLX problems.

There is another class of pulsed NNs in which the positions of the pulses have certain meaning [16]. In addition, another type of simultaneous perturbation method called time difference simultaneous perturbation [17] has been proposed. Using a similar approach to that described in this paper, we could easily implement these types of networks as well.

REFERENCES

- [1] B. J. Sheu and J. Choi, *Neural Information Processing and VLSI*. Boston, MA: Kluwer, 1995.
- [2] J. C. Spall, "A stochastic approximation technique for generating maximum likelihood parameter estimates," in *Proc. American Control Conf.*, 1987, pp. 1161–1167.
- [3] —, "Multivariable stochastic approximation using a simultaneous perturbation gradient approximation," *IEEE Trans. Automat. Contr.*, vol. 37, pp. 332–341, 1992.
- [4] J. Alespector, R. Meir, B. Yuhua, A. Jayakumar, and D. Lippe, "A parallel gradient descent method for learning in analog VLSI neural networks," in *Advances in Neural Information Processing Systems 5*, S. J. Hanson, J. D. Cowan, and C. Lee, Eds. San Mateo, CA: Morgan Kaufmann, 1993, pp. 836–844.
- [5] G. Cauwenberghs, "A fast stochastic error-descent algorithm for supervised learning and optimization," in *Advances in Neural Information Processing Systems 5*, S. J. Hanson, J. D. Cowan, and C. Lee, Eds. San Mateo, CA: Morgan Kaufmann, 1993, pp. 244–251.
- [6] Y. Maeda and Y. Kanata, "Learning rules for recurrent neural networks using perturbation and their application to neuro-control" (in Japanese), *Trans. Inst. Electr. Eng. Japan*, vol. 113-C, no. 6, pp. 402–408, 1993.
- [7] —, "A learning rule of neural networks for neuro-controller," in *Proc. 1995 World Congr. Neural Networks*, vol. 2, 1995, pp. II-402–II-405.
- [8] Y. Maeda and R. J. P. de Figueiredo, "Learning rules for neuro-controller via simultaneous perturbation," *IEEE Trans. Neural Networks*, vol. 8, pp. 1119–1130, Sept. 1997.
- [9] Y. Maeda, H. Hirano, and Y. Kanata, "A learning rule of neural networks via simultaneous perturbation and its hardware implementation," *Neural Networks*, vol. 8, no. 2, pp. 251–259, 1995.
- [10] G. Cauwenberghs, "An analog VLSI recurrent neural network learning a continuous-time trajectory," *IEEE Trans. Neural Networks*, vol. 7, pp. 346–361, Mar. 1996.
- [11] E. I. El-Masry, H. Yang, and M. A. Yakout, "Implementations of artificial neural networks using current-mode pulse width modulation technique," *IEEE Trans. Neural Networks*, vol. 8, pp. 532–548, May 1997.
- [12] A. F. Murray, D. D. Corso, and L. Tarassenko, "Pulse-stream VLSI neural networks mixing analog and digital techniques," *IEEE Trans. Neural Networks*, vol. 2, pp. 193–204, Mar. 1991.
- [13] Y. Maeda, A. Nakazawa, and Y. Kanata, "Hardware implementation of a pulse density neural network using simultaneous perturbation learning rule," *Analog Integrated Circuits and Signal Processing*, vol. 18, no. 2, pp. 1–10, 1999.
- [14] H. Hikawa, "Frequency-based multilayer neural networks with on-chip learning and enhanced neuron characteristics," *IEEE Trans. Neural Networks*, vol. 10, pp. 545–553, May 1999.
- [15] S. Hauck, "The role of FPGA's in reprogrammable systems," *Proc. IEEE*, vol. 86, pp. 615–638, Apr. 1998.
- [16] W. Maass and C. M. Bishop, Eds., *Pulsed Neural Networks*. Cambridge, MA: MIT Press, 1998.
- [17] Y. Maeda, "Time difference simultaneous perturbation," *Electron. Lett.*, vol. 32, no. 11, pp. 1016–1018, 1996.