

The pulse density representation is sometimes used for neural systems. It is also called different names such as stochastic logic[5,6], pulse-mode neuron[7], bit-stream neuron[8], frequency-based network[9] and so on[10]. We also know spiking neural networks, which realize central pattern generator, and their hardware implementation[11].

Totally, the pulse density type of neural networks has attractive properties. We can summarize the merits of pulse density neural networks as follows;

- Biological analogy
- Invulnerability to noisy environment
- Analogue system realized by digital system
- Ease of system design assisted by the EDA technique
- Economizing circuit area

The analogy to biological neural systems is intriguing. A part of our nerve systems are employing the pulse density scheme. Especially, it is well-known that nerve impulses are transmitted from retina to visual cortex. These systems are invulnerable to noisy conditions. Limited noises do not result in serious malfunction of the overall system. Moreover, we can handle quantized analog quantities, based on the digital technology that is relatively effortless to design and implement. Recent EDA technique can assist trouble-free design of overall system. In addition, pulse system simplifies arithmetic circuit. For example, multipliers are realized by the AND operation. This results in economization of the circuit design.

The learning capability is the most important factor of artificial neural network systems. Even when we consider hardware implementation of neural network systems, realizing the learning ability is essential. In other words, it is crucial to adopt suitable learning scheme for hardware implementation.

Eguchi et al. reported a pulse density neural system[12]. They used the back-propagation method as a learning rule. However, the learning scheme requires complicated circuitry, since the learning rule uses derivatives of an error function to update weights. This results in larger area on chip or difficulty in circuit design. Particularly, it seems impossible to realize large scale of neural network system.

On the other hand, the simultaneous perturbation learning scheme[3] is an option. Because of its simplicity, the learning rule is suitable for hardware implementation[3,13,14]. The learning scheme is applicable to recurrent neural networks as well[15].

Moreover, if we adopt pulse density representation, we can take advantage of the simultaneous perturbation effectively[16].

The combination of the pulse density system and the simultaneous perturbation learning rule results in an easy configuration which can be implemented in the hardware neural network system. In our hardware implementation, pulse density is used to represent the weight values, outputs and inputs.

2.2 Analog recurrent neural network

Compared with the feed forward neural networks, recurrent neural networks(RNNs) have inherent and interesting properties such as dynamics. As a typical example of RNN, let us think about Hopfield neural network(HNN)[17]. HNNs are used to store patterns or to solve combinatorial optimization problems like the traveling salesman problem. For these problems, the weights in the network are typically determined by patterns to be memorized based on Hebbian learning rule[18] or an energy function based on the problem. If our patterns are analogue or if we cannot find a proper energy function, it will be impossible to apply these techniques to find the optimal weight values of the network.

Hebbian learning is widely used for the HNN type of neural networks including the bidirectional associative memories. However, this learning rule can cope with only binary values; +1 and -1 for bipolar representation, +1 and 0 for unipolar one.

Since the HNNs can handle analog quantities as same as the other ordinary neural networks, it is important to contrive a new learning scheme and suitable representation of analog quantities.

The simultaneous perturbation learning scheme is applicable to the analog problems[13]. Moreover, when we combined this with the pulse density representation of analog quantities, the simultaneous perturbation learning scheme increases in value.

2.3 Learning via simultaneous perturbation

When we use a RNN for a specific purpose, we need to determine the proper values of the weights in the RNN. That is, the so-called learning of RNNs is necessary.

In many applications of RNNs, we know the ideal output for the network. Using this information, we can evaluate how well the network performs. Such an evaluation function gives us a clue for optimizing the weights of the network. Of course, like Hebbian learning, we can determine the weight value through off-line learning. However, on-line learning scheme for RNNs is interesting.

Thus we consider a recursive learning scheme for RNNs. Then back-propagation through time(BPTT) is typical example. In order to use the BPTT, the error quantity must propagate through time from a stable state to an initial state. This process is so complicated. It seems difficult to use such a method directly, because it takes a long time to compute the modifying quantities corresponding to all weights. At the same time, it seems practically difficult to realize the learning mechanism as a hardware system. In addition, BPTT is not applicable to the situation that RNNs learn an oscillatory outputs or trajectory solution.

On the other hand, the simultaneous perturbation scheme is proposed[19] and it is shown that the learning scheme is suitable for the learning of NNs and their hardware implementation[3,13-16]. The simultaneous perturbation optimization method requires only values of an evaluation function as mentioned. If we know the evaluation of a stable state, we can obtain the modifying quantities of all weights of the network without complicated error propagation through time.

The simultaneous perturbation learning rule for recurrent neural networks is described as follows;

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

$$\Delta \mathbf{w}(t) = \frac{J(\mathbf{w}(t) + c\mathbf{s}(t)) - J(\mathbf{w}(t))}{c} \mathbf{s}(t) \quad (2)$$

Where, $\mathbf{w}(t)$ denotes the weight vector of a network at the t -th iteration. α is a positive constant, c is the magnitude of the perturbation. $\Delta \mathbf{w}(t)$ represents a common quantity to obtain the modifying vector for all the weights. $\mathbf{s}(t)$ denotes a sign vector whose element $s^i(t)$ is 1 or -1 with zero mean. The sign of $s^i(t)$ are randomly determined. That is, $E(s^i(t)) = 0$, moreover, $s^i(t_1)$ and $s^j(t_2)$ are independent with respect to different components i and j , and different time t_1 and t_2 . Where, E denotes the expectation. $J(\mathbf{w})$ denotes an error or an evaluation function, for example, defined by outputs of neurons in a stable state and a pattern to be embedded.

We can summarize the advantages of the simultaneous perturbation learning rule for NNs as follows;

- Applicability to RNNs
- Error back propagation through time is not necessary
- It is simple
- Applicability to analogue problems
- Applicability to oscillatory solutions or trajectory learning

- An energy function is not necessary

3 FPGA Implementation

3.1 Design Background

There are some ways of realizing hardware RNNs with learning capability. For example, C.Lehmann et al. reported a VLSI system for a HNN with on-chip learning [20]. Also in our previous research, the FPGA implementation based on digital circuit design technology was used to realize the HNN[13]. On the other hand, we are considering a FPGA implementation of a pulse density RNN with a recursive learning capability using the simultaneous perturbation method. Neurons in the network have connected each other.

We adopted VHDL (VHSIC Hardware Description Language) in basic circuit design for FPGA. The design result by VHDL is configured on MU200—SX60 board (Mitsubishi Electric Micro-Computer Application Software Co., Ltd.) with EP1S60F1020C6 (Altera) (see Fig.1). This FPGA contains 57,120 LEs with 5,215,104 bit user memory. The number of LE prescribes the scale of the neural network such as number of neurons.

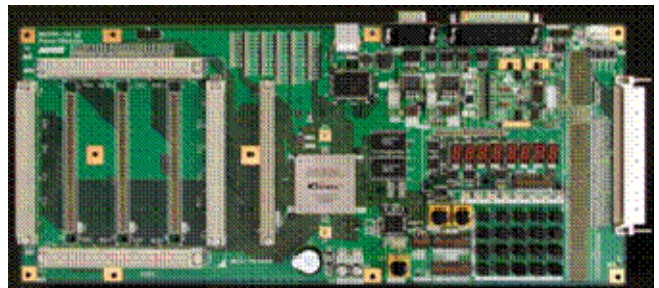


Fig.1 FPGA board MU200—SX60

Visual Elite (Summit) is used for the basic design. Synplify Pro(Synplicity) carried out the logical synthesis for the VHDL. Finally, Quartus II (Altera) is used for wiring. The result is sent to the FPGA through BYTE-BLASTER-II.

In this work we adopt pulse density representation with sign bit. Maximum number of pulses is 255. Addition to this, sign bit shows that the signal is positive or negative. As a result, we can handle values in the range $[-1(=-255/255) + 1(=+255/255)]$. As a result, resolution of the values is $1/255$.

3.2 Configuration

Fig.2 shows overall configuration of the RNN system. The system consists of three units of the RNN unit, the weight modification unit and the learning unit.

The RNN unit realizes ordinary operation of the RNN to obtain a stable state based on certain weights. The weight modification unit updates all weights used in the RNN unit. The learning unit generates modifying quantities for all the weights using the simultaneous perturbation learning rule.

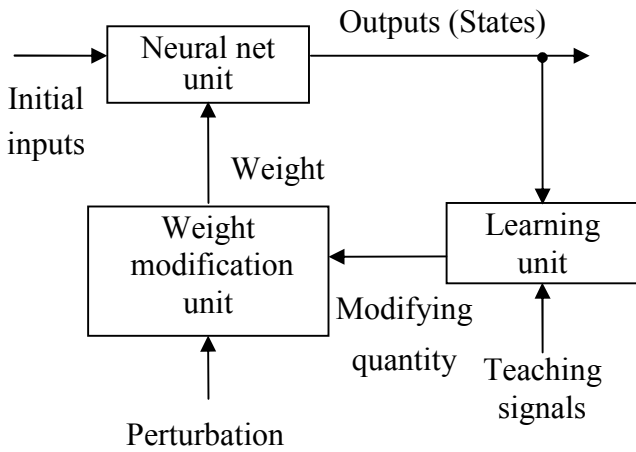


Fig.2 System configuration.

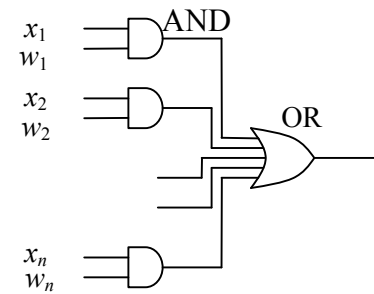
3.3 Neural network unit

Pulse density scheme enable the RNN simple, since weighted sum of neuron operations are realized by AND and OR operations shown in Fig.3(a). AND operations can realize multiplication of inputs and the corresponding weights. Next OR operation is used for addition of the previous results. The RNN unit consists of the neurons.

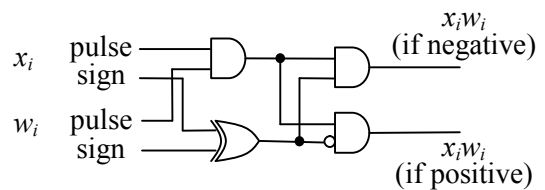
We designed the values of weights and outputs by pulse train with sign signals. Therefore, multiplication is replaced by Fig.3(b). Then results of the circuit are connected up/down counter. If the result is positive, we find pulses in the lower output. The line is connected to up counter. If the result is negative, we find pulse train in the upper line. The line is connected to down counter. As a whole, this circuit realizes a multiplication of pulses with sign signal. We can implement multiplication and summing up by the simple circuit configuration. Similar idea is used in many works handling pulse-based operation[6,9].

Input-output characteristic or activation function is realized by saturation of pulse train. Even if multiplication of weight value and input is too large, the summing result is limited by the maximum number of pulses in a certain specified period. Between the upper and lower limitation, the result is

linearly produced. As a result, a piecewise linear function with a certain saturation is realized as the input-output characteristic of a neuron.



(a) Unsigned pulse operation



(b) Signed pulse operation

Fig.3 Pulse operation.

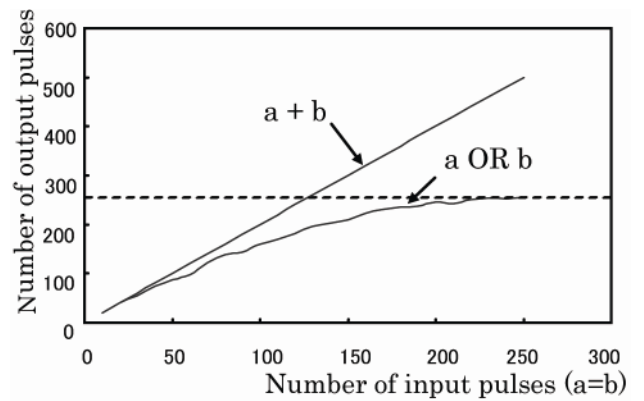


Fig.4 Input-output characteristic.

Actually, collisions of some pulses are happen, so that some overlapped pulses are neglected. As a result, we obtain a rounded curve as the characteristic as shown in Fig.4. Fig.4 shows an actual addition result in pulse expression. The horizontal axis denotes a number of pulses as inputs a and b. The vertical axis shows observed pulse number as an output. Therefore, twice of the horizontal axis is theoretical result for the vertical axis. However, in Fig.4, the actual result is rounded for the linear theoretical one, and has saturation at 255.

Without any specific operation, we can realize sigmoid-like property using pulse expression. In many analog hardware neural network systems, there are many reports for realizing the sigmoid function and this is one of the points to correctly realize the function of overall neural system. From this point of view, pulse expression is beneficial.

Combining this circuit, we can implement the operation of RNN. Repeating the operation of the circuit, the system obtains a stable state of the network. This procedure is relatively easy.

With pulse representation, the unit is so simple and consumes smaller circuit size than ordinary binary representation as a RNN system. When we have stable outputs, these outputs signals are sent to the learning unit to realize the recursive modification of the weights.

3.4 Learning unit

The learning unit achieves the so-called learning process using the simultaneous perturbation and sends the basic modifying quantity to the weight modification unit, 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 operations of the RNN. When the RNN arrive at a stable state, the state, namely output is probed by the learning unit. Based on the output and its corresponding desired output, the learning unit generates modifying quantities for all the weights via the simultaneous perturbation.

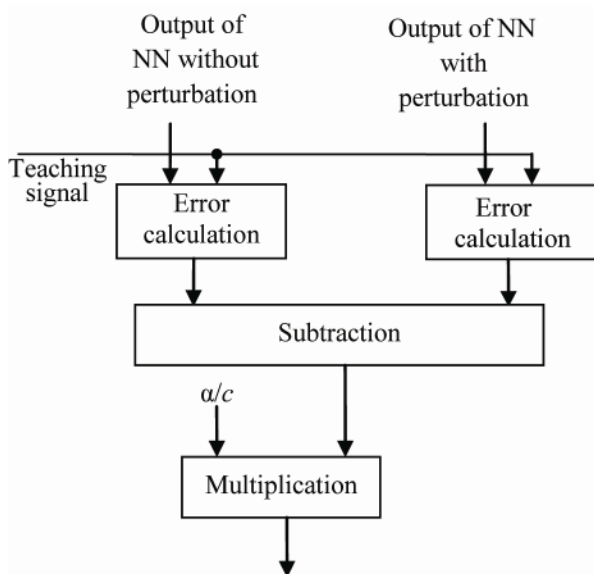


Fig.5 Learning unit.

Error defined by the final states of the RNN and

their ideal ones is as follows in this work. This absolute error makes the system simple, because a simple counter can realize this calculation.

$$J(\mathbf{w}(t)) = \sum_j |O_j(t) - T_j(t)| \quad (3)$$

Where, O_j and T_j are final output of RNN and their teaching signal, respectively. j is index for neuron number in the network.

In the learning process, the following quantity is common for all weights.

$$\alpha \Delta \mathbf{w}(t) = \alpha \frac{J(\mathbf{w}(t) + c\mathbf{s}(t)) - J(\mathbf{w}(t))}{c} \quad (4)$$

To implement the above quantity easily we set α/c as 1, 1/2, 1/4, 1/8, 1/16, 1/32, 1/64, 1/128, 1/256 or 1/512. This process is realized by bit shift. Therefore, the operation becomes very easy and simplifies the circuit design.

3.5 Weight modification unit

Fig.6 shows flowchart of the weight modification unit. This unit controls the weight values of RNN. In this system, perturbed operation of the RNN and ordinary operation are required to generate basic quantity for weights modification.

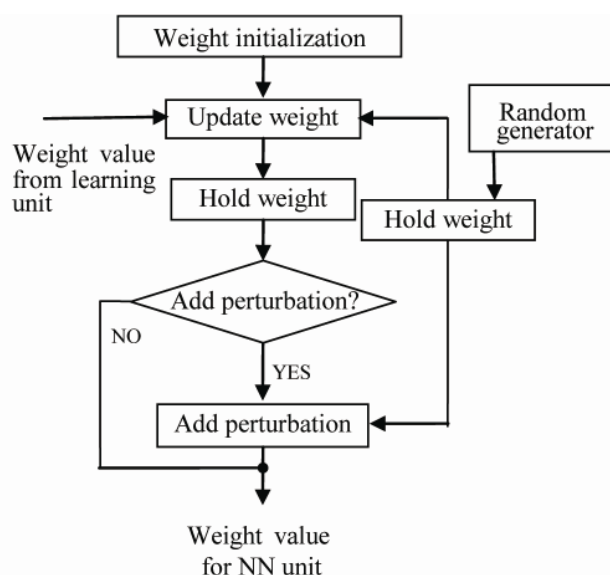


Fig.6 Flowchart of the weight modification unit.

3.6 Other technical factors

It is generally difficult for hardware system to produce random numbers. To generate random number, the linear feedback shift register is used.

All system requires control unit. This RNN system also requires clock signal and many timing signals. This system contains the circuit for this purpose. The clock frequency is 19.5MHz for actual

operation of the FPGA system.

Interface for LED display and ten-key, or output for measurement is not essential from academia point of view but important for practical experiment. This function was also designed in this system.

4 HNN System

HNN is a typical example of recurrent NNs proposed by Hopfield. HNN realizes auto-association in which incomplete or corrupted initial pattern is recovered. Output of HNN is determined as follows;

$$O(t+1) = f(WO(t)) \tag{5}$$

Where O is state or output vector for neurons of HNN. f is a output-input characteristic such as the sigmoid function. t denotes iteration. For a certain weight matrix W whose diagonal elements are zero, repeating the calculation of Eq.(5) gives a stable state. This is basic operation of HNN.

Now, we fabricate the HNN system using FPGA. We consider examples for HNN with 25 neurons. The HNN learns the following two examples. The perturbation c is $7/255$ and the learning coefficient α is $(7/255) \times (1/26)$. This setting is empirically determined through preliminary experiments.

4.1 Character problem

The first example is to memorize a character shown in the following figure of capital A. Numbers written in Fig.7 shows neuron number.

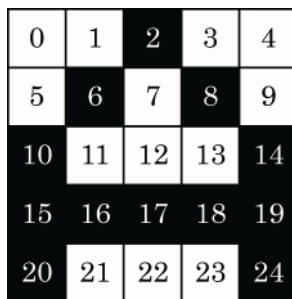


Fig.7 Character A.

Black pixels represent value of $1(=255/255)$, that is, corresponding neuron has to produce 255 pulses for a certain interval and white pixels are $-1(=-255/255)$, that is, corresponding neuron has to produce 255 pulses with negative sign signal. This is basically a binary problem.

Configuration result is shown in Table 1. As in the table, 45% of logic elements are used. Some pins are used to observe operation of the system.

Table 1 Configuration result for EP1S60F1020C6.

Flow Status	Successful - Thu Feb 03 17:20:18 2005
Revision Name	PDM_HNN25_SP_SYS
Top-level Entity Name	PDM_HNN25_SP_SYS
Family	Stratix
Device	EP1S60F1020C6
Total logic elements	26,222 / 57,120 (45 %)
Total pins	78 / 782 (9 %)
Total memory bits	0 / 5,215,104 (0 %)
DSP block 9-bit elements	1 / 144 (<1 %)
Total PLLs	0 / 12 (0 %)
Total DLLs	0 / 2 (0 %)

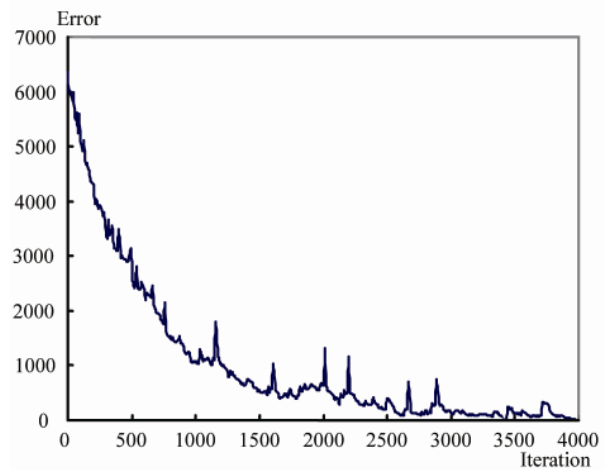


Fig.8 Change of error in FPGA system.

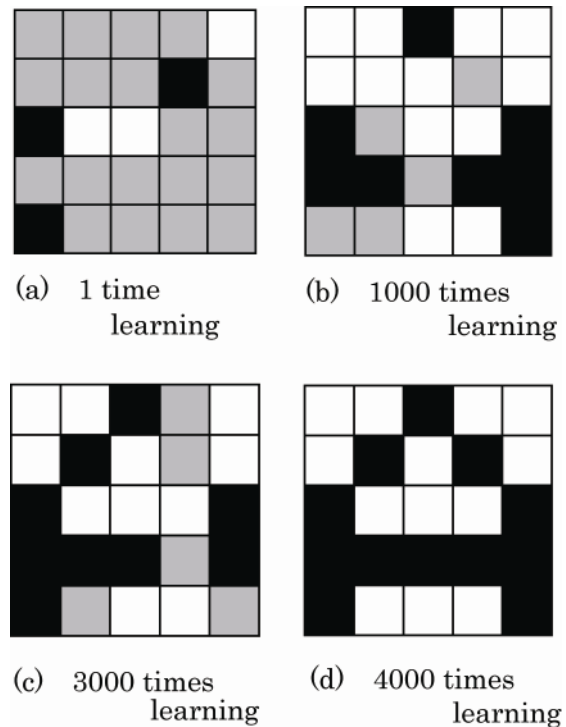


Fig.9 Recalled patterns.

Fig.8 shows change of the error defined by Eq.(4) in learning process. As learning proceeds, the error

decreases. After about 4000 times learning, the system perfectly learns the character of A. Since the simultaneous perturbation learning rule is based on a stochastic gradient method, the learning curve does not monotonously decrease.

Fig.9 shows change of recalled patterns. In this figure, black pixel and white one mean that outputs of the corresponding neuron are positive and negative respectively. If the output of the HNN is unstable, the pixel is depicted in grey.

From Fig.8 and Fig.9, we can see that the system learns the pattern A. After about 4000 learning, the system recalls the pattern perfectly from a corrupted initial state.

4.2 Sinusoidal pattern

Next we consider an analog problem. The HNN system must learn sinusoidal wave in the range [-1(=-255/255) +1(=255/255)].

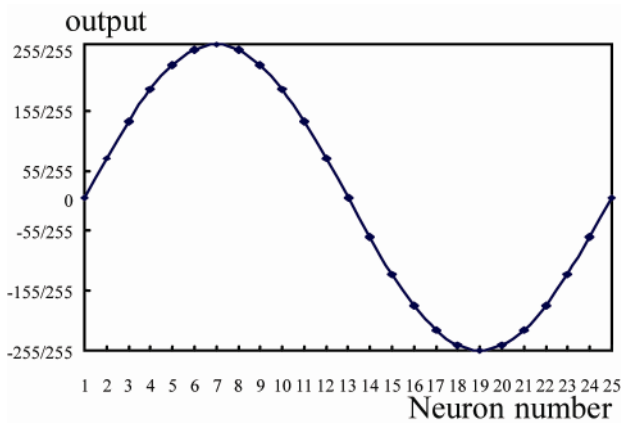


Fig.10 Teaching pattern for each neurons.

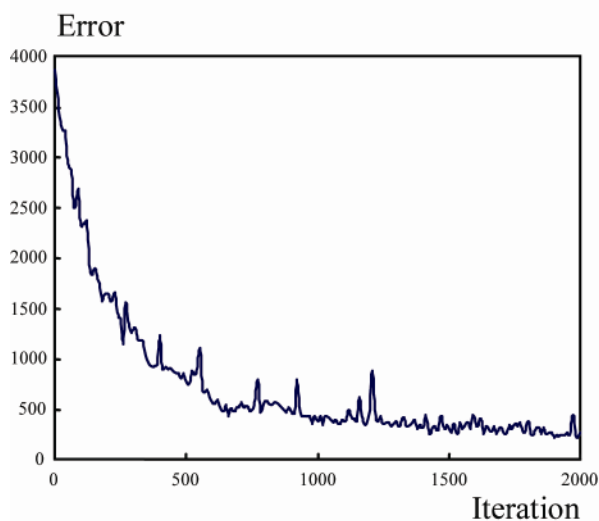


Fig.11 Change of error in FPGA system.

The HNN contains 25 neurons which memorize analog values shown in Fig.10 respectively. For

example, the first neuron has to output no pulse because the desired output is 0, the seventh neuron has to produce 255 pulses because the output is 1 and so on.

Change of the error function in learning process is depicted in Fig.11. Error decreases as iteration proceeds. This figure shows that the system learns the sinusoidal pattern as iteration proceeds.

5 BAM System

Bidirectional associative memory proposed by Kosko in 1988 is an extension of Hopfield neural network. The network realizes the hetero-associative memory in which recalled patterns are different from triggering patterns as HNN can realize auto-association.

BAMs consist of two layers. Based on certain weight values and initial states of neurons in the A layer, inputs of neurons in the B layer are calculated. And input-output characteristic determines outputs of neurons in the B layer. Next these outputs of the B layer and weights become inputs of neurons in the A layer. Then outputs of the A layer determined. That is, the following calculation is carried out repeatedly.

$$O^B = f(WO^A) \tag{6}$$

$$O^A = f(WO^B) \tag{7}$$

Where O^A and O^B are state or output vector of neurons in the A layer and the B layer, respectively. f is a output-input characteristic. For a certain weight matrix W , repeating the operations give a stable state of neurons in the network. Then we can obtain certain pair of output pattern of the A layer and the B layer. This is basic operation of BAM. Evaluation function is defined as follows;

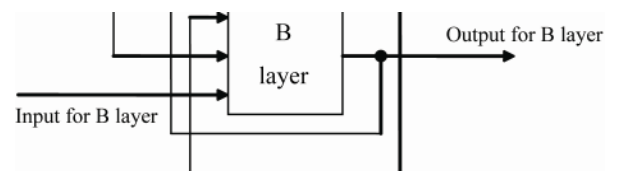


Fig.12 BAM unit.

$$J(w(t)) = \sum_{j \in A \text{ layer}} |O_j^A(t) - T_j^A(t)| + \sum_{j \in B \text{ layer}} |O_j^B(t) - T_j^B(t)| \tag{8}$$

Where, O_j^A and O_j^B denote the i -th elements of \mathbf{O}^A and \mathbf{O}^B respectively. T means their corresponding teaching outputs.

Now, we fabricate the BAM system with 3 neurons in the two layers respectively. The BAM learns the following two examples. The perturbation and the learning coefficient are determined shown in the following tables through preliminary experiments. The BAM unit is shown in Fig.12.

5.1 Binary pattern

The first example for BAM is to memorize a pair of the following simple pattern shown in Fig.13. A layer and B layer must learn three black and three white pattern. Black and white pixels mean that outputs of the pixels denoted $1(=255/255)$ or $-1(=0/255)$ respectively in stable state. Parameters in the learning and learning values as the teaching pattern are shown in Table 2.



(a) A layer (b) B layer

Fig.13 Teaching pattern for BAM.

Table 2 Parameters and learning values.

Parameters	
Perturbation c	0.0156 (4/255)
Learning coefficient α	0.0078 (4/255 × 1/2)
Learning pattern for the layer A	
Learning value for neuron 1	1(=255 / 255)
Learning value for neuron 2	1(=255 / 255)
Learning value for neuron 3	1(=255 / 255)
Learning pattern for the layer B	
Learning value for neuron 1	-1(=255 / 255)
Learning value for neuron 2	-1(=255 / 255)
Learning value for neuron 3	-1(=255 / 255)

Change of the error is shown in Fig.14. After about 110 learning, error decreases suddenly. About 200 learning is enough to obtain sufficient recall for the desired pattern.

Fig.15 shows recall results under learning process. In these results, grey pixels mean that the output of the neuron is not stable. After 200 learning, stable

outputs are equal to the desired ones shown in Fig.15(d).

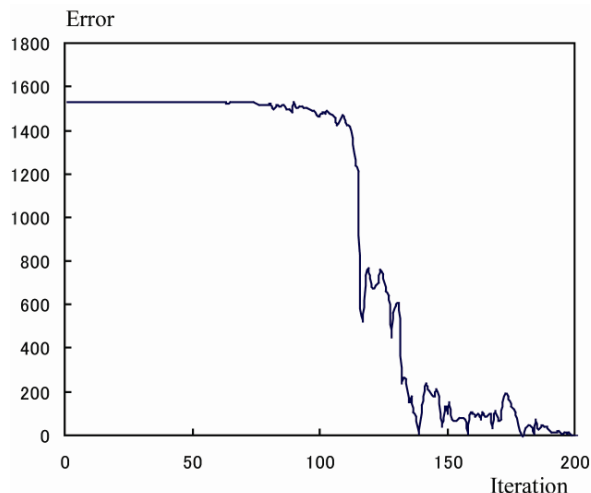


Fig.14 Change of error in FPGA system.

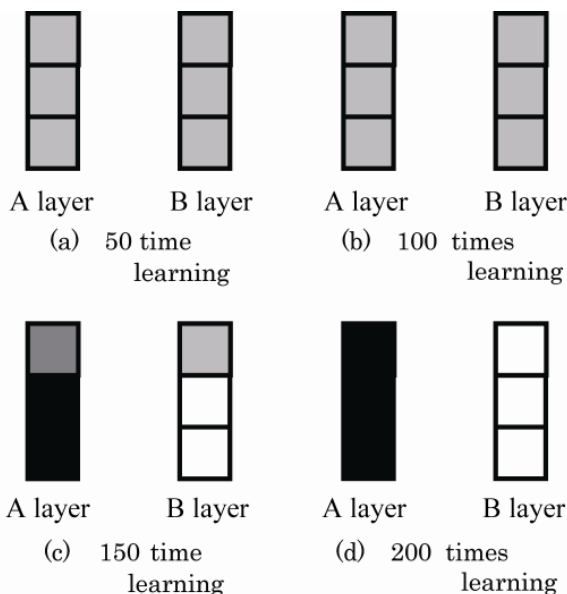


Fig.15 Change of error in FPGA system.

5.2 Analogue pattern

Now, we consider an analogue problem in which target outputs are analogue. When we handle analogue problems, we cannot use Hebbian learning to determine weight values of the network. Then recursive learning scheme is important. Even for the analogue problems, we can apply the simultaneous perturbation learning scheme as same as digital or binary problems.

We handle the analogue quantities shown in Table 3. Six neurons have to learn analogue value such as $0.59(\approx 150/255)$ or $0.78(\approx 200/255)$ and so on. Used parameters are also written in the table.

Table 3 Learning pattern

Parameters	
Perturbation c	0.0549 (14/255)
Learning coefficient α	0.0137(14/255 \times 1/4)
Learning pattern for the layer A	
Learning value for neuron 1	150 / 255
Learning value for neuron 2	200 / 255
Learning value for neuron 3	250 / 255
Learning pattern for the layer B	
Learning value for neuron 1	-150 / 255
Learning value for neuron 2	-200 / 255
Learning value for neuron 3	-250 / 255

Change of the error is shown in Fig.16. After about 50 learning, error decreases suddenly. About 200 learning, the BAM could recall the desired pattern. The BAM system learns the analogue problem as well.

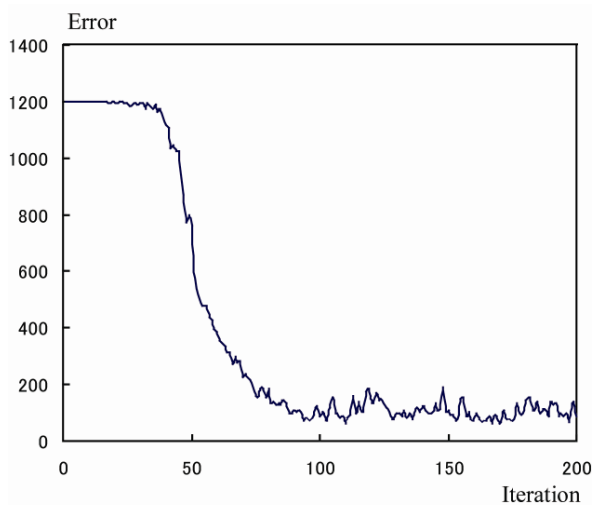


Fig.16 Change of error in FPGA system.

6 Conclusion

In this paper, we described FPGA RNN systems with learning capability based on pulse expression. HNN and BAM were handled as actual systems. The simultaneous perturbation method is adopted as the learning scheme. We could effectively realize a useful hardware learning mechanism. Moreover, we confirm a feasibility of recurrent neural networks using the simultaneous perturbation learning method.

This system could handle both analog and digital problems as well.

Acknowledgement

This work is financially supported by MEXT KAKENHI (No.19500198) of Japan, Academic Frontier Project, and the Intelligence system technology and kansei information processing research group, ORDIST, Kansai University.

References:

- [1] M. Zheng, M. Tarbouchi, D. Bouchard, J. Dunfield, FPGA Implementation of a Neural Network Control System for a Helicopter, WSEAS TRANSACTIONS on SYSTEMS, Vol. 5, 2006, pp.1748-1751.
- [2] C. Lee, C. Lin, FPGA Implementation of a Fuzzy Cellular Neural Network for Image Processing, WSEAS TRANSACTIONS on CIRCUITS and SYSTEMS, Vol. 6, 2007, pp.414-419.
- [3] 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, 1995, pp. 251-259.
- [4] H. S. Ng and K. P. Lam, Analog and digital FPGA implementation of BRIN for optimization problems, IEEE Trans. Neural Networks, Vol. 14, 2003, pp.1413-1425.
- [5] Y. Kondo and Y. Sawada, Functional abilities of a stochastic logic neural network, IEEE Trans. Neural Networks, Vol. 3, 1993, pp. 434-443.
- [6] S. Sato, K. Nemoto, S. Akimoto, M. Kinjo, and K. Nakajima, Implementation of a new neurochip using stochastic logic, IEEE Trans. Neural Networks, Vol. 14, 2003, pp.1122-1127.
- [7] H. Hikawa, A new digital pulse-mode neuron with adjustable activation function, IEEE Trans. Neural Networks, Vol. 14, 2003, pp. 236-242.
- [8] D. Braendler, T. Hendtlass, and P. O'Donoghue, Deterministic bit-stream digital neurons, IEEE Trans. Neural Networks, Vol. 13, 2002, pp.1514-1525.
- [9] H. Hikawa, Frequency-based multilayer neural network with on-chip learning and enhanced neuron characteristics, IEEE Trans. Neural Networks, vol. 10, 1999, pp.545-553.
- [10] M. Martincigh and A. Abramo, A new architecture for digital stochastic pulse-mode neurons based on the voting circuit, IEEE Trans. Neural Networks, Vol. 16, 2005, pp.1685-1693.
- [11] L. Bako, S. Brassai, Spiking Neural Networks Built in FPGAs: Fully Parallel Implementations, WSEAS TRANSACTIONS on CIRCUITS and SYSTEMS, Vol. 5, 2006, pp.346-353.

- [12] H. Eguchi, T. Furuta, H. Horiguchi and S. Oteki, Neuron model utilizing pulse density with learning circuits, IEICE Technical Report, vol.90, 1990, pp.63-70. (in Japanese)
- [13] Y. Maeda and M. Wakamura : Simultaneous perturbation learning rule for recurrent neural networks and its FPGA implementation, IEEE Trans. Neural Networks, Vol. 16, No. 6, 2005, pp.1664-1672.
- [14] 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, 1999, pp. 1–10.
- [15] Y. Maeda and M. Wakamura : Bidirectional associative memory with learning capability using simultaneous perturbation, Neurocomputing, Vol. 69, 2005, pp.182-197.
- [16] Y. Maeda and T. Tada : FPGA implementation of a pulse density neural network with learning ability using simultaneous perturbation, IEEE Trans. Neural Networks, Vol. 14, No. 3, 2003, pp.688-695.
- [17] J. J. Hopfield, Neurons with graded response have collective computational properties like those of two-state neuron, in Proc. Nat. Acad. Sci., Vol. 81, 1984, pp. 3088–3092.
- [18] B. Kosko, Neural Networks and Fuzzy Systems. Englewood Cliffs, NJ: Prentice-Hall, 1992.
- [19] J. C. Spall, Multivariable stochastic approximation using a simultaneous perturbation gradient approximation, IEEE Trans. Autom. Control, Vol.37, 1992, pp. 332–341.
- [20] C. Lehmann, M. Viredaz, and F. Blayo, A generic systolic array building block for neural networks with on-chip learning, IEEE Trans. Neural Networks, Vol. 4, 1993, pp. 400- 407.