Implementation of Simplified Multilayer Neural Networks with On-chip Learning

Hiroomi Hikawa
Oita University
hikawa@csis.oita-u.ac.jp

ABSTRACT

In this paper, a new digital architecture of Multilayer Neural Network (MNN) with on-chip learning is proposed. Proposed MNN is designed to have no multiply operation for efficient hardware implementation. The absence of the multiplier makes the circuit size small, thus the proposed MNN is suitable for massively parallel VLSI implementation.

To provide the on-chip learning ability, the back-propagation algorithm is modified to have no multiply operation, and the algorithm is implemented with pulse-mode operation. Further, a tri-state function is used as the activate function of neurons so that the multipliers in forward path is replaced by a combination of shift and logical AND operations, which are easily realized by digital circuits.

The proposed MNN is implemented on a field programmable gate array (FPGA) and tested. To verify the feasibility of the proposed MNN in the larger application, the MNN design is tested using a pattern recognition problem by computer simulations.

1. Introduction

A salient feature of Artificial Neural Networks (ANN) is their learning ability. Size and real-time considerations show that on-chip learning is necessary for a large range of applications. A flexible digital design is preferred to more compact analog or optical realizations. Another advantage of using digital approach is that it can use state-of-the-art VLSI and ULSI implementation techniques. One of the major problems in the implementation of ANN with digital architecture is the presence of multipliers, which affects the circuit size and performance. Neural networks without multiplier have been proposed [1][2].

In this paper a multiplier-less Multilayer Neural Network (MNN) with on-chip learning is proposed. To provide the on-chip learning ability, the back-propagation algorithm is modified to have no multiply operation and the algorithm is implemented with pulse-mode operation. As a result, required hardware for the on-chip learning is very simple and small. To remove the multipliers in forward path, the proposed MNN employs a tri-state function as the activation function of neurons. Use of the tri-state function replaces the multiplier by a combination of shift and logical AND operations, which are easily realized by digital circuits. A bit-serial communication and bit-serial arithmetic operations are used to reduce the overall circuit size, and communication links.

The proposed MNN is implemented on a field programmable gate array (FPGA), and its feasibility is tested by experiments and computer simulations.

2. Multilayer Neural Network

MNN is one of the most widely used neural network model. The MNN may consist of one or more hidden layers of neurons between the input and the output layers. Each neuron is connected to all neurons of the two adjacent layers via synaptic weights.

The operation of the MNN is divided into two phases, i.e., learning phase and retrieving phase. During the learning phase, weights are adjusted to perform a particular application. The learning phase consists of forward path and backward path. In the forward path, the output of the network is calculated form input data, and the learning algorithm is performed during the backward path. In the retrieving phase, the same operation as the forward path is performed.

2.1. Forward path

During the forward path, data from neurons of a lower layer is propagated forward to neurons of an upper layer via feed-forward connection network. Let $o_k^{(s)}$ denote the output of k-th neuron of the s-th layer, and the layers are numbered from 0 to M,

0-7803-2768-3/95/$4.00

then the computation performed by each neuron is

$$H_k^{(s)} = \sum_{j=1}^{N_{s-1}} w_{kj}^{(s)} o_j^{(s-1)} + \theta_k^{(s)} \quad (1)$$

$$o_k^{(s)} = f(H_k^{(s)}) \quad (2)$$

where $H_k^{(s)}$ is the weighted sum of the k-th neurons of the s-th layer and $w_{kj}^{(s)}$ is the synaptic weight. The output $o_k^{(s)}$ of the neuron is obtained by computing an activation function $f(\cdot)$ on the weighted sum.

2.2. Backward path

Training algorithm is performed in the backward path. The back-propagation algorithm [3] is the most widely used training algorithm in MNN. Criterion of the learning algorithm is to minimize the least-square-error between the teacher value and the actual output value. At the very beginning of the learning phase, the forward path is executed to obtain the ouput response of the input training pattern. Then error between the teacher value and the actual ouput value is propagated in backward, and the error is used to update the weight values.

The back-propagation algorithm is expressed by,

$$\sigma_k^{(s)} = \begin{cases} t_k - o_k^{(s)} & s = M \\ \sum_{j=1}^{N_{s+1}} w_{kj}^{(s+1)} \delta_j^{(s+1)} & \\ & s = 1, \cdots M-1 \end{cases} \quad (3)$$

$$\delta_k^{(s)} = \sigma_k^{(s)} f'(H_k^{(s)}) \qquad s = 1, \cdots, M \quad (4)$$

$$\Delta w_{kj}^{(s)} = \eta \delta_k^{(s)} o_j^{(s-1)} \quad (5)$$
$$k = 1, \cdots N_s \qquad j = 1, \cdots N_{s-1}$$

3. Multilayer Neural Networks without Multipliers

3.1. Forward path without multiplier

The proposed MNN uses a tri-state function $g(\cdot)$ as the activation function:

$$g(x) = \begin{cases} 0 & x < -TH \\ 1/2 & -TH \le x \le TH \\ 1 & x > TH \end{cases} \quad (6)$$

As $g(\cdot)$ takes only three values the multiplication of $w_{kj}^{(s)} o_j^{(s-1)}$ in (1) is realized with a combination of shift and logical AND circuitry, which is much simpler than the multiplier. Also the function generator for $g(\cdot)$ can be easily implemented with comparators. To reduce the communication links between neurons and synapse unit in the MNN, weighted neuron outputs are transferred through serial communication links. Consequently, the calculation of the weighted sum is carried out by bit-serial adders.

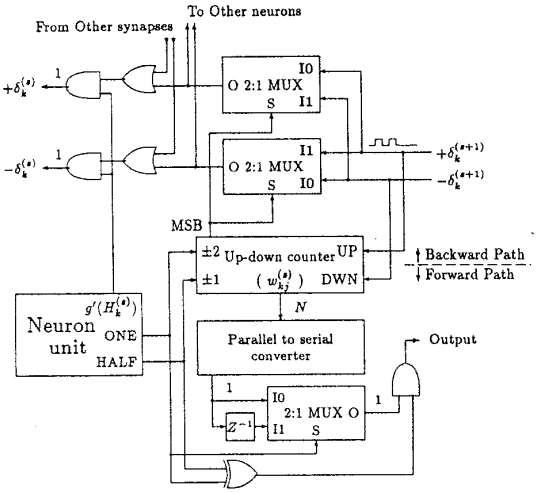

Fig. 1: Synapse unit

Table: 1: Shift operation

$o_k^{(s)}$	operation
1	one-clock delay
1/2	no delay is applied
0	transmission is disabled

The block diagram of the synapse unit that weights the neuron output is depicted in Figure 1. Lower portion of the figure is used in the forward path. Weight value is in N-bit, fixed-point format, and is stored in an up-down counter so that it can be easily changed by applying pulses to the up-down clock inputs. The content of the counter is then converted to a bit-serial data and is sent to the next layer from LSB first. The shift operation is performed by changing the timing of the transmission. For example, if $o_k^{(s)}$ is 1/2, corresponding weight is sent in advance by one-clock period so that the data is shifted to the right by one-bit, which is equivalent to a division by two. The shift operations are summarized in Table 1.

The block diagram of the modified neuron unit which calculates the weighted sum is shown in Fig. 2. In the neuron unit, all weighted outputs of the neurons in the lower layer are summed up by bit-serial adders, then the weighted sum is fed to comparators which are used as function generators for $g(\cdot)$ and its derivative function $g'(\cdot)$ (See Table 2).

3.2. Modified back-propagation algorithm

The back propagation model was not designed for efficient hardware implementation. Therefore, the

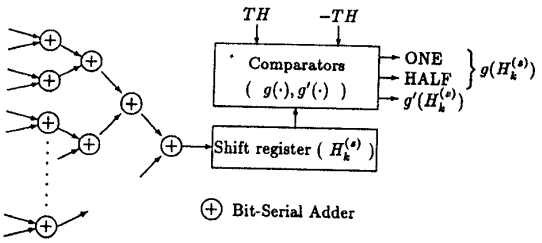

Fig. 2: Neuron unit

Table: 2: Output of the neuron

	$g'(H_k^{(s)})$ ONE	HALF	$g'(H_k^{(s)})$
$H_k^{(s)} < -TH$	0	0	0
$-TH \leq H_k^{(s)} \leq TH$	0	1	1
$TH < H_k^{(s)}$	1	0	0

algorithm is modified for effective hardware implementation. Especially, the modification is focused on elimination of multipliers. The upper half of the synapse unit depicted in Fig. 1 is the hardware implementation of the modified back-propagation algorithm, which operates during the backward path.

For the effective hardware implementation, the error terms $\sigma_k^{(s)}$, and $\delta_k^{(s)}$ in (3) are represented by pulse signals. These signals are used to update the up-down counters that contain the weight values. The number of the pulse is proportional to the absolute value of the error, and these pulses are propagated through $+\delta$ line if the error is positive, otherwise $-\delta$ line is used. When the absolute value of the error is one, it is represented by two pulses, 0.5 is represented by a single pulse and no pulse is present when the error is 0. As $\sigma_k^{(2)}$ (error at each output) takes only five different values, i.e., ± 1, ± 0.5 or 0, maximum number of error pulses is two. To calculate $w_{kj}^{(s+1)}\delta_j^{(s+1)}$ in (3), instead of the actual weight value, the sign bit (MSB) of the weight is multiplied with the error term $\delta_j^{(s+1)}$ [4]. As the sign bit takes two values (zero for positive, one for negative), the multiplication is significantly simplified. The circuit to perform the operation is two multiplexers which exchange two signals $+\delta$ and $-\delta$ when the MSB of the weight is one.

Function $f'(\cdot)$ in (4) is the derivative of the activation function. Following function is used for $f'(\cdot)$ in the proposed MNN:

$$g'(x) = \begin{cases} 0 & x < -TH \\ 1 & -TH \leq x \leq TH \\ 0 & x > TH \end{cases} \qquad (7)$$

Table: 3: Function of the Up-down counter

INPUT UP	DWN	±2	±1	OUTPUT $Q_0 \sim Q_{N-1}$
X	X	0	0	No change
↑	0	0	1	Inc. by one
↑	0	1	0	Inc. by two
0	↑	0	1	Dec. by one
0	↑	1	0	Dec. by two
X	X	1	1	No change

The multiplication $\sigma_k^{(s)}g'(H_k^{(s)})$ in (4) can be realized by two AND gates because the function $g'(\cdot)$ is a binary function. The multiplication, $\delta_k^{(s)}o_j^{(s-1)}$ in (6), and updating of the weights are performed as follows; each time the $\delta_k^{(s)}$ pulse is applied to the up-down counter, the content of the counter is increased or decreased by the size of $o_j^{(s-1)} \times 2$. Special up-down counter whose increment and decrement size can be changed is devised to implement this operation. The operation of the counter is depicted in Tab. 3,

4. Experiments and Simulations

4.1. Implementation on FPGA

The proposed MNN has been implemented on a Field Programmable Gate Array (FPGA), and tested. The configuration of the experiment is depicted in Fig. 3. The experimental MNN has two input-node, three hidden-node, two output nodes, and 17 connections. Training circuit that generates the error signals as well as other necessary signals for the MNN has been fabricated on another FPGA. Third node in the input layer and fourth node in the hidden layer always generate one output so that the synaptic weights connected to neurons act as the offset θ of the neurons. In Fig. 4 the signals of the MNN during the learning process is displayed.

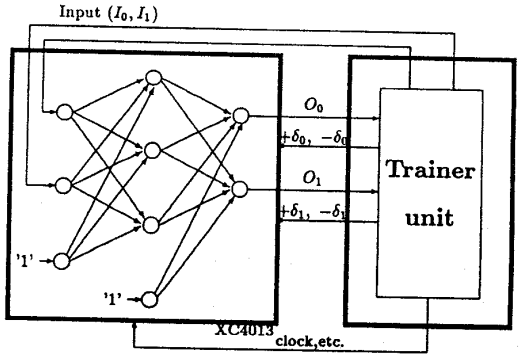

Fig. 3: Configuration of the experiment

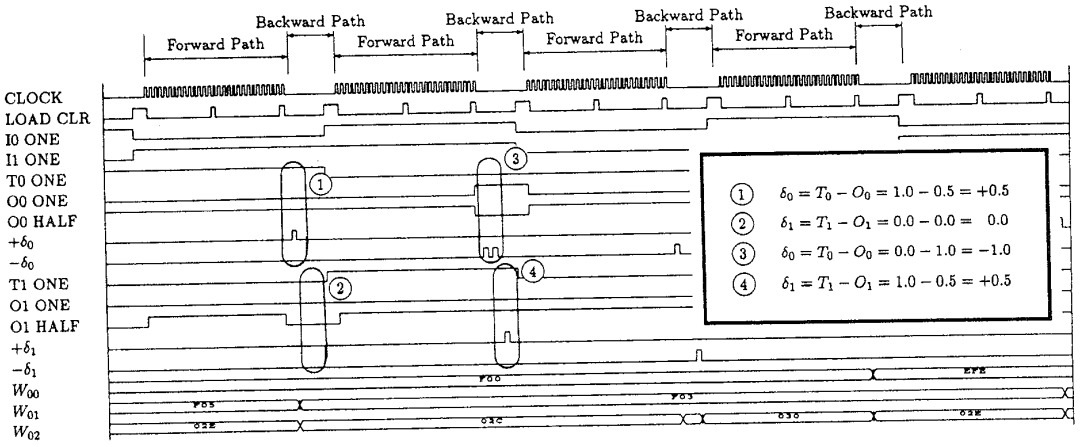

Fig. 4: Signals in the proposed MNN learning logic functions.

First, input data I_0, I_1 are given by the trainer unit, and the forward path is processed. In the backward path, the error signals are generated according to the output (O_0 and O_1) resulted from the forward path and teaching data (T_0 and T_1). The error signals δ_0 (error at the output node 0) and δ_1 (error at the output node 1) are generated in tern. In the figure, a single pulse is generated for $+\delta_0$ because $\delta_0 = +0.5$, and no pulse is generated for δ_1 because $\delta_1 = 0.0$. These processes continue and the error disappears eventually, as shown in Fig. 5. The figure shows that the proposed MNN has been successfully trained to perform two logical functions (exclusive-OR and AND). The experimental MNN has 12-bit fixed-point format for the weight value, and TH is set to 256. The experimental network runs with 25 MHz clock. With the 25 MHz clock rate, It takes 1.16 μs (29 clocks) for forward calculation, which is equivalent to the performance of 14.6×10^6 connections per second (CPS). For both forward and backward operation, it requires 1.56 μs (39 clocks), and delivers 10.9×10^6 connection updates per second (CUP). Note that all neuron units and synapse units are connected in 'fully parallel' and all units work concurrently. Hence, these performance figures are proportional to the size of networks.

4.2. Performance analysis with computer simulations

To verify the feasibility of the proposed MNN in the larger application, the MNN design has been tested using a pattern recognition problem by computer simulations. The same configuration of the MNN that is tested by the experiments discussed in the previous section, has been simulated by this simulation program. The experimental results and the simulation results are exactly the same. Thus this simulation program is validated by the experiments and its result is very reliable.

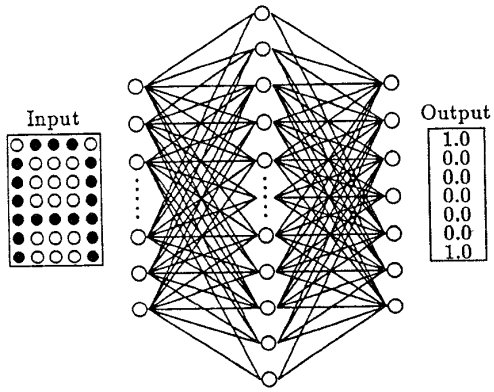

Fig. 6: Configuration of the Character Recognition Application

In the pattern recognition problem, the inputs of the MNN are a seven by five array that is a pixel representation of a single character, and the output is 7-bit ASCII code. Fig. 6 shows the configuration of the system. The MNN has been trained to recognize 110 different character patterns corresponding to 7-bit ASCII codes. Fig. 7 shows the learning behavior of the proposed MNN. This figure shows that the output error is steadily reduced and the MNN is trained successfully. However, with fewer hidden neurons (less than 30), it becomes difficult for the tri-state MNN to be trained. The tri-state MNN with hidden neurons less than 25 has convergence problem while the neural networks with sigmoid neurons doesn't.

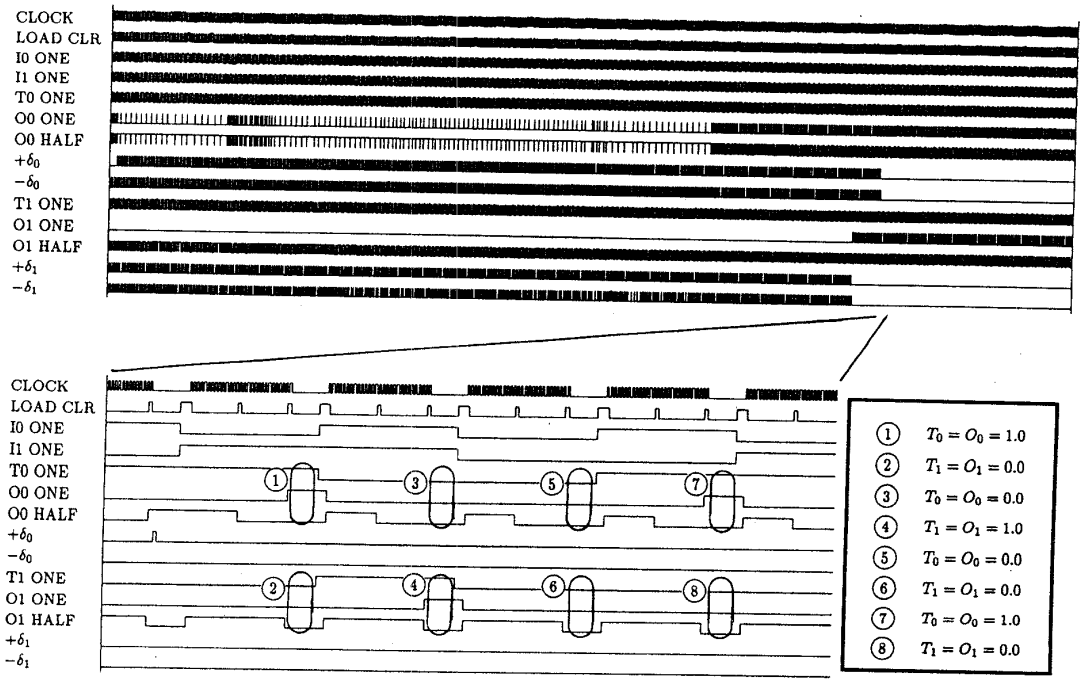

Fig. 5: Signals after the learning

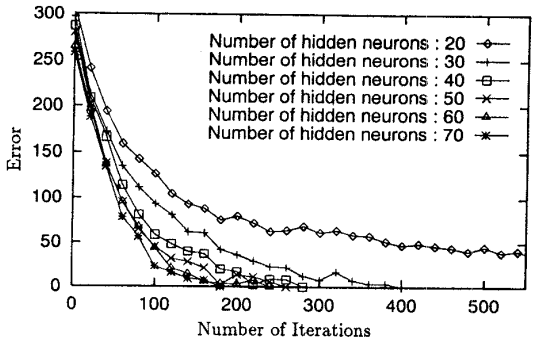

Fig. 7: Learning behavior of the proposed MNN in the pattern recognition application

5. Conclusions

In this paper, hardware implementation of multilayer neural network with on-chip learning has been discussed. The back-propagation algorithm is modified to have no multiplier and the algorithm was implemented with pulse mode operation. Using the tri-state function as the activating function, multiplications are replaced by much simpler operations, such as shift and logical AND operations. The proposed MNN was implemented on an FPGA, and the feasibility of the proposed MNN and the modified back-propagation were verified by both experiments and computer simulations. Experimental circuit runs with 25 MHz clock, and delivers 14.6×10^6 CPS and 10.9×10^6 CUP. The behavior of the MNN used in pattern recognition was studied by computer simulations and the results show that the proposed network can be used for larger applications.

Simple structure of the proposed MNN leads to a massive parallel and flexible network architecture, which is well suited for VLSI implementation. The greatest potential of neural nets remains in the high-speed processing that could be provided through massively parallel VLSI implementations.

References

[1] B. A. White, M. I. Elmasry, "The digi-neocognitron: a digital neocognitron neural network model for VLSI," *IEEE Trans. on Neural Networks*, Vol.3, pp.73-85, January 1992.

[2] M. Marchesi, G. Orlandi, F. Piazza and A. Uncini, "Fast Neural Networks Without Multipliers," *IEEE Trans. on Neural Networks*, Vol.4, No.1, Jan. 1993.

[3] R. P. Lipmann, "An introduction to computing with neural nets," *IEEE Acoustics, Speech, and Signal Processing Magazine*, pp. 4-22, April 1987.

[4] T. Baker and D. Hammerstrom, "Characterization of Artificial Neural Networks Algorithms," *Proc. of 1989 ISCAS*, pp.78-81, 1989.