Realizing TDNN for Word Recognition on a Wavefront Toroidal Mesh-array Neurocomputer

Hong Jeong, Cha-Gyun Jeong, and Myung-Won Kim

Abstract

In this paper, we propose a scheme that maps the time-delay neural network(TDNN) into the neurocomputer called EMIND-II which has the wavefront toroidal mesh-array structure. This neurocomputer is scalable, consists of many timeshared virtual neurons, is equipped with programmable on-chip learning, and is versatile for building many types of neural networks. Also we define the programming model of this array and derive the parallel algorithms about TDNN for the proposed neurocomputer EMIND-II. In addition, the computational complexities for the parallel and serial algorithms are compared. Finally, we introduce an application of this neurocomputer to word recognition.

I. Introduction

Attempts have been made to implement digital neurocomputer by means of the network of processors each of which performs basic functions such as multiplication, addition for synapse computation and inter-processor communication for signal transfer between neurons [1], [2], [3], [4], [5]. Since late 1980's, lots of neural networks have appeared in the market. Many products are, however, issued in the form of more or less software packages simulating the mathematical functions of artificial neural networks. Others have appeared as board-level products built out of conventional DSP chips. In fact, only a few genuine neural chips have been actually produced. Because of its difficulty in usage and limited capabilities, these chips are not widely spread. Some hardware products may be found in the survey [6].

In light of this, early in 1992, we implemented a digital neurocomputer called *ETRI Machine Imitating Neuro Dynamics* (EMIND), which was built from *Digital Neural Processor* (DNP) as a unit processor [7], [8]. In 1994, we have fabricated an improved neural chip called DNP-II and thereby built the EMIND-II neurocomputer [9].

Basically, the neurocomputer is a scalable parallel architecture based on asynchronous inter-processor communication for simulating large scale neural networks. The EMIND-II is considered a systolic array incorporated with data-driven computing. In the network each PE is independently programmed and it communicates with its neighbors in an asynchronous manner. Unlike the SIMD the wavefront array needs no global clock for controlling the entire network of processors and each PE can be clocked differently. Thus, it is possible to construct massive parallel network consisting of many PEs. In addition, each processor can deal with multiple neurons due to the time-sharing capability, and therefore the machine can emulate very large virtual networks. The digital VLSI circuit makes it easy to interface with a host computer via the conventional parallel/serial ports. Additionally, the on-chip learning capability and torus connection between boundary neurons allow the implementation of many types of neural networks such as *Multilayer Perceptron* MLP [10], *Time Delay Neural Network*(TDNN) [11], Hopfield networks [12], and *Self-organizing* Map (SOFM) [13]. In this paper, we shall focus only on the TDNN.

The purpose of this paper is to develop a possibly general scheme that drives the neurocomputer to emulate a TDNN such that a simple change of TDNN parameters such as weights, number of layers and neurons, and learning parameters may lead to a variety of applications. Mainly, the scheme explains how to load the initial parameters from the host, how to drive the neurocomputer to execute the forward and backward passes for classification and learning purposes,

Manuscript received September 12, 1995; accepted October 11, 1995.

H. Jeong and C.G.Jeong are with Department of Electrical Engineering, Postech, Pohang, Korea.

M.W. Kim is with School of Computing, Soongsil Univ., Seoul, Korea.

and how to store the result back to the host.

To achieve this goal, we must first define a programming model of the neurocomputer and the TDNN structure. Based upon these models, we can then investigate how to program the network to behave like TDNN. In this context, this paper addresses the following topics. First, as a minimal knowledge for the development of TDNN scheme, section II introduces the internal structure of the machine viewed by the programmer. The TDNN structure and the serial algorithm are discussed in the following section section III. After then, section IV derives parallel algorithms of the neurocomputer for executing the TDNN recognition and learning procedures. This parallel algorithm is compared with the serial algorithm for TDNN in terms of the computational complexities in section V. Also, an application to word recognition is discussed.

II. Architecture of the Toroidal Mesh Array

To begin with, we introduce the architecture of EMIND-II and the neural chip, DNP-II in a top down order. A detailed discussion on the architecture is dealt in [9].

1. *The EMIND-II Architecture*

The EMIND-II is a slave processor that must be supervised by the host computer. It is the role of the host to load the programs and data into the internal memory of the slave processor and read the result from the slave processor. One of the useful features of our architecture is the capability of parallel memory loading. When the data is loaded into the array, it is written into columns (rows) of Processor Elements (PEs) in parallel, like high bandwidth memory with multiple buses. Once initiated, the machine simply repeats the sequential execution of the stored programs, possibly reading the input data, processing it, and sending the results to the host.

The actual EMIND-II is a single board processor that consists of 8 x 8 DNP-IIs. Since each DNP-II contains 4 PEs, it is equivalent to 16 x 16 PEs. Notwithstanding the particular size and parameters of the actual implementation, the algorithm is developed in a general setting. Let us assume that EMIND-II consists of M x M processors, where M is an integer greater than 2. Fig. 1 illustrates this array processor for the case of M = 16.

For convenience's sake, each $PE_{i,j}$ is identified by its site (i,j) in the grid plane and thus the architecture of the machine can be conveniently represented by the matrix $\{PE_{i,j} \mid i \in [0, M-1], j \in [0, M-1]\}$.

It is convenient to partition PEs into four separate blocks according to their roles: $\{PE_{i,j} \mid i \in [0, M-1], j \in [0, M-2]\}$, $\{PE_{0,j} \mid j \in [0, M-2]\}$, $\{PE_{i,M-1} \mid i \in [1, M-1]\}$, and $PE_{0,M-1}$. The central role of the first block is to compute the sum-of-products. The next two blocks are used as either parallel inputs or outputs of the neural network. The last block can be used as an interface with the host whenever a serial communication is needed.

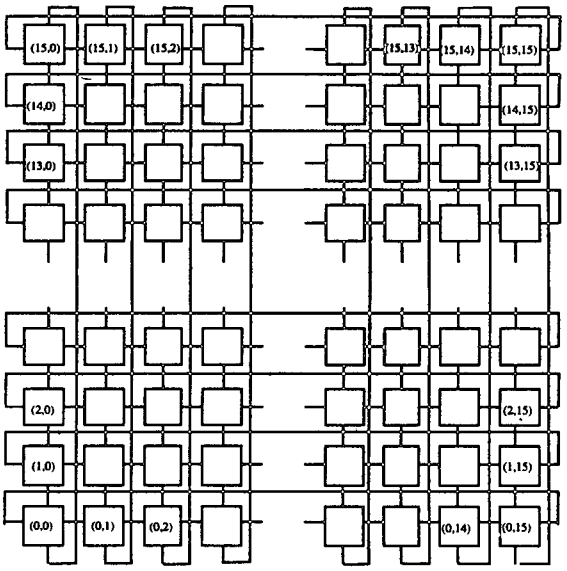


Fig. 1. Network structure of EMIND-II with M=16.

The elements on the opposite boundaries are connected one-to-one configuring a toroidal structure. The 2-dimensional toroidal grid is a convenient architecture for mapping many artificial neural networks. If the processors are equipped with handshaking protocols, the size of this array will be easily expanded without danger of clock skewing along the long mismatched data paths. Also the digital circuit allows an easy interface to the digital computer. Therefore, the parallel interface for loading and storing data from/to the host computer can be easily accomplished. Finally, the capability of time-sharing amounts to a huge network which contains many multiples of M^2 PEs.

2. *The DNP-II Architecture*

Many *Digital Signal Processors* (DSPs) and TRANSPUTERS have been used as the PEs for neural network simulators. Such processors should not necessarily be cost-effective for computing neural networks because they are originally designed for different types of computational tasks from neural computation. It is highly desirable to design a processor well tailored to neural processing. Processor architectures for digital neural processing have been implemented or proposed including the X1 of Adaptive Solutions [1], the MA16 of Siemens [14], and the SPERT of the University of California at Berkeley [15]. However,

they are processors having SIMD (or systolic) architecture.

The DNP-II is a unit processor for our EMIND-II neurocomputer; it is the second generation of the DNP [7], [8] which has been designed for our first neurocomputer EMIND. The DNP-II is considered similar to the DSP in architecture and function, however, its functions are simpler and more tailored to neural computation than the DSP. It also facilitates efficient asynchronous communication exploiting the regularity and locality of neural computation. Its key features include 0.8 μ CMOS standard technology, 40 MHz operating clock, a 16x16 bit parallel pipelined multiplier, four-way 16 bit parallelasynchronous communication, 256 words (16 bit a word) of program memory, 128 words of input memory, and 512 words of weight memory. It is also devised with instruction prefetch and subroutine call capability.

Actually, the chip contains 4 PEs, each having identical structures as shown in Fig. 2.

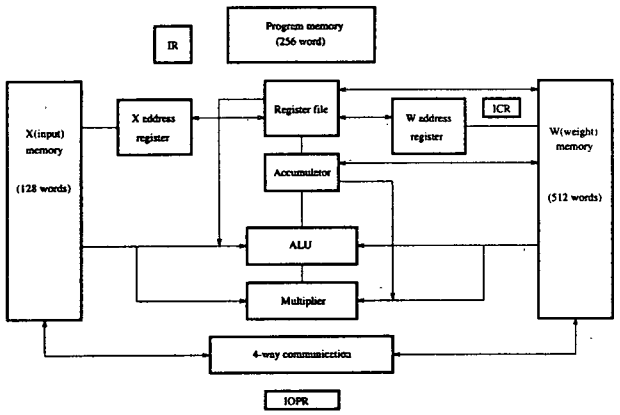


Fig. 2. Internal structure of DNP-II.

It consists of four major functional building blocks: memory block, control block, arithmetic block,and I/O interface block. The memory block consists of 512 words of data memory and 128 words of input memory. The data memory is used for storing weights, intermediate results, tables, and initial parameters. It also has an adder for address generation aiming at flexible data access. There are four sets of increment/decrement counters for address generation. Each instruction involving memory access needs to specify one of these registers. The input memory is used to temporarily store data read in through the communication channel for recycling them.

The control block consists of a separate program memory of 256 words of 16 bits and an instruction decoder. It contains a counter which specifies the number of repetitions to execute instructions. The arithmetic block consists of a multiplier, an arithmetic/logic unit (ALU), and general purpose registers. The ALU performs addition, subtraction, shifting, and bitwise logic functions. The communication block consists of four asynchronous parallel I/O communication units. The block also has a special register (IOPR) which contains four pair of I/O port numbers. Each instruction involving I/O communication should specify one of these pairs.

The DNP-II takes a single clock cycle for executing most instructions but two clock cycles for I/O communication. The instruction set of the DNP-II is optimized for various neural network models and learning algorithms. The DNP-II operates basically in 16 bit, fixed-point integers. It also has a three-stage pipelined instruction of "multiply and accumulate" which multiplies an input value from the input memory and a weight from the weight memory, then accumulates the result into the accumulator.

III. The TDNN

As is well known, this network is a multi-layer feed-forward neural network that can be trained to classify specific spectral structures within consecutive frames of speech [11] into some disjoint classes. To meet the need of an extensive computational load for executing the recognition and training, one has to rely on a hardware solution.

1. The TDNN architecture

Fig. 3 outlines the typical TDNN architecture.

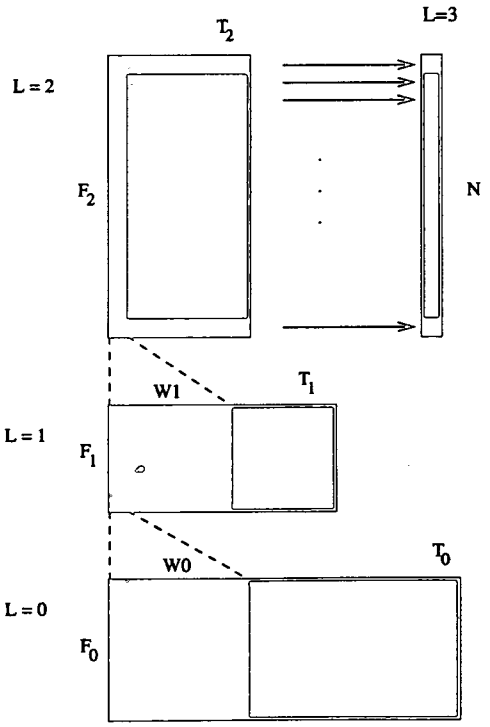


Fig. 3. A typical paradigm for TDNN structure.

It consists of 4 layers labeled as L = 0,...,3. The input frame consists of an F_0 x T_0 matrix. Each time the first layer takes a small matrix of the F_0 x W_0 window from the F_0 x T_0 input frame and produces an F_1 x 1 output vector. Consequently, the F_0 x T_0 input frame generates the F_1 x T_1 output frame. This output frame is further supplied as an input of the next layer. In this manner, the F_1 x T_1 input frame is transformed to the N x T_2 output frame. As a final stage, the output layer yields the N x 1 vector as a result of this network. It is expected that the output vector contains some normalized scores that indicate how the input pattern is similar to the reference patterns. It is well known that this score is in fact *a posteriori* probability [16]. Also one of the best known learning algorithms for this network is the *Error Back Propagation* (EBP) algorithm [10].

Some parameters depend upon others. For example, T_1 = T_0 - W_0 + 1, and T_2 = T_0 - W_0 - W_1 + 2. Therefore, there are only 6 independent variables, F_0, F_1, W_0, W_1, T_0, and N.

2. *Serial Algorithms for Recognition and Training*

Let $x^{(l)}_{i,j}$ denote the output of (i,j) neuron at level $l \in [0, 3]$. Fig. 4 illustrates an example.

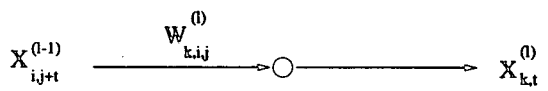


Fig. 4. A neuron $x^{(l)}_{,t}$ is connected to $x^{(l-1)}_{i,j+t}$ by an weight $w^{(l)}_{k,i,j}$.

Note that the weight is translation invariant. It follows that the output of layer $l \in [0, 3]$ is represented by the matrix, $\{x^{(l)}_{i,j} | i \in [0, F_l-1], j \in [0, T_l-1]\}$. Also assume that $w^{(l)}_{k,i,j}$ is the connection strength between the (i,j) neuron at level l - 1 and the k neuron at level l. Finally, assume that $\phi^{(l)}_k$ is the threshold of this neuron.

The notations allow the following equations for the recognition phase of TDNN.

$$\begin{cases} x^{(1)}_{k,t} = f\left(\sum_{j=0}^{W_0-1}\sum_{i=0}^{F_0-1} w^{(1)}_{k,i,j} x^{(0)}_{i,j+t} - \phi^{(1)}_k\right), \text{ for } k\in[0,F_1-1], t\in[0,T_1-1], \\ x^{(2)}_{k,t} = f\left(\sum_{j=0}^{W_1-1}\sum_{i=0}^{F_1-1} w^{(2)}_{k,i,j} x^{(1)}_{i,j+t} - \phi^{(2)}_k\right), \text{ for } k\in[0,N-1], t\in[0,T_2-1], \\ O_i = \sum_{t=0}^{T_2-1} (x^{(2)}_{i,t})^2, \text{ for } i\in[0,N-1] \end{cases} \tag{1}$$

where O denotes output. One can easily derive that the number of neurons, free weights, and multiplications are respectively

$$F_1T_1 + NT_2, \tag{2}$$

$$F_1(F_0W_0+1) + N(F_1W_1+1), \tag{3}$$

$$F_0W_0F_1T_1 + F_1W_1NT_2. \tag{4}$$

In the learning phase, we must first compute

$$\begin{cases} \delta^{(2)}_{i,t} = (O_i - R_i)x^{(2)}_{i,t} f'(x^{(2)}_{i,t}), \\ \triangle w^{(2)}_{i,j,k} = \alpha\frac{1}{T_2}\sum_{t=0}^{T_2-1} \delta^{(2)}_{i,t} x^{(1)}_{j,k+t} + \eta\triangle\ w^{(2)}_{i,j,k}, \\ \triangle \phi^{(2)}_i = \alpha\frac{1}{T_2}\sum_{t=0}^{T_2-1} \delta^{(2)}_{i,t} + \eta\triangle\ \phi^{(2)}_i, \\ (i\in[0,N-1], j\in[0,F_1-1], k\in[0,T_1-1], t\in[0,T_2-1]). \end{cases} \tag{5}$$

Here $f'(\cdot)$ denotes the derivative, R is the target output for learning. Also η and α are constants. $\triangle W$ and $\triangle \phi$ are the weight incremental values at the previous step. Following these operations, one must compute

$$\begin{cases} \delta^{(1)}_{i,t} = f'(x^{(1)}_{i,t})\sum_{j=0}^{N-1} w^{(1)}_{j,i,t}\ \delta^{(2)}_{j,t}, \\ \triangle w^{(1)}_{i,j,k} = \alpha\frac{1}{T_1}\sum_{t=0}^{T_1-1} \delta^{(1)}_{i,t} x^{(0)}_{j,k+t} + \eta\triangle\ w^{(1)}_{i,j,k}, \\ \triangle \phi^{(1)}_1 = \alpha\frac{1}{T_1}\sum_{t=0}^{T_1-1} \delta^{(1)}_{i,t} + \eta\triangle\ \phi^{(1)}_i, \\ (i\in[0,F_1-1], j\in[0,F_0-1], k\in[0,T_0-1], t\in[0,T_1-1]). \end{cases} \tag{6}$$

As a result, the forward pass, delta and weight update take

$$(F_0W_0F_1T_1 + F_1W_1NT_2) + (NT_2 + F_1T_1N) + (NF_1T_1T_2 + F_1F_0T_0T_1) \tag{7}$$

multiplications. Here the first term is the quantity required by the forward pass. On the other hand, the second and the third terms denote the amount of multiplications needed by the delta and weight update, respectively. The relationships are summarized in Table 1. This with other properties will be discussed in section V in detail.

IV. Parallel Algorithms for Emulating TDNN by the EMIND-II

Having established the hardware and TDNN models, we are faced with the question of what scheme to realize TDNN on the hardware. In this section, we first derive the recognition algorithm and then the learning algorithm.

1. *Parallel Recognition Algorithm*

Rules (1), (5) and (6) are serial algorithms that completely describe the recognition and learning phases of the model TDNN. From now on, these rules are further converted in accordance with the hardware structures of EMIND-II. Without loss of generality, let us assume F_0 = 15, F_1 = 15, W_0 = 8, W_1 = 7, T_0 = 20, and N = 20 for simplicity.

Some of the snapshot view is illustrated in Fig. 5.
As we see in Fig. 5(a), the input data is loaded into $PE_{0,j}$. Next in Fig. 5(b), the inputs propagate unaltered in the upper direction. Simultaneously the partial sum-of-products are propagated in the right direction. As a result, the final result of the first layer is accumulated in $PE_{i,15}$. The second layer is realized in Fig. 5(c) with the result stored in $PE_{0,j}$.

According to this algorithm, $PE_{i,0}$ and $PE_{0,j}$ must contain the look-up-table for $f'(\cdot)$. All the other PEs except $PE_{15,15}$ must contain the connection strengths and thresholds for the two layers.

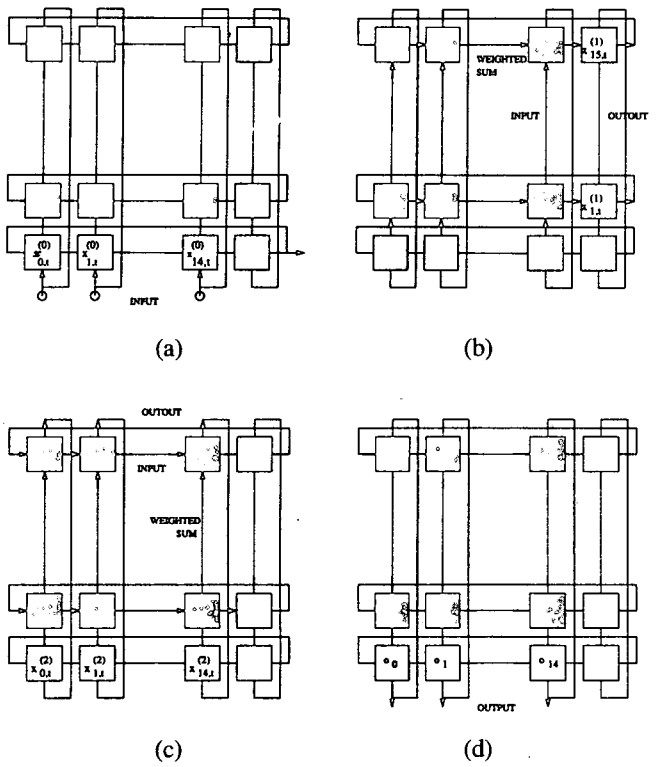


Fig. 5. Forward data flow for the recognition phase of TDNN: (a) the input, (b) the 1st layer, (c) the 2nd layer, and (d) the output.

The operations can be described as follows. The first step is to initialize $PE_{0,j}$. Let $X_{i,j}^{(l,t)}$ denote the data stored at $PE_{i,j}$ representing the data at t $\in$ [0, T_l-1] at level l. Then $PE_{0,j}$ must be loaded with the input data:

$$Step\ 1: X_{0,i}^{(0,t)} = x_{i,t}^{(0)}, \text{ for } t\in[0, T_0-1], i\in[0, F_0-1]. \quad (8)$$

Likewise $X_{i,15}^{(1,t)}$ denotes the output of level 1 to be stored in $PE_{i,15}$. Then, we have for the 1st layer

$$Step\ 2: \begin{cases} X_{i,j}^{(1,t)} = X_{i,j-1}^{(1,t)} + \sum_{k=0}^{W_0-1} w_{i,j,k}^{(1)} X_{0,j}^{(0,k+t)}, \\ X_{i,15}^{(1,t)} = f(X_{i,14}^{(1,t)}), \text{ for } i\in[1, F_1], j\in[0, F_0-1], t\in[0, T_1-1]. \end{cases} \quad (9)$$

In the next phase, the result in $PE_{i,15}$ becomes the input of the next layer. In Fig. 5(b), the input and output propagate respectively in the right and upper directions in PEs. The results of this layer accumulate in $PE_{0,j}$.

$$Step\ 3: \begin{cases} X_{i,j}^{(2,t)} = X_{i-1,j}^{(2,t)} + \sum_{k=0}^{W_1-1} w_{j,i,k}^{(2)} X_{i,15}^{(1,k+t)}, \\ X_{0,j}^{(2,t)} = f(X_{15,j}^{(2,t)}), \text{ for } i\in[1, F_1], j\in[0, N-1], t\in[0, T_2-1]. \end{cases} \quad (10)$$

Finally in Fig. 5(c), vector sum must be performed for the data in $PE_{0,j}$ to produce a vector featuring pattern class and store the results there to be transferred to the host computer.

$$Step\ 4: \begin{cases} X_{0,j} = \sum_{t=0}^{T_2-1} X_{0,j}^{(2,t)}, \\ o_j = X_{0,j}, \text{ for } j\in[0, N-1]. \end{cases} \quad (11)$$

2. *Parallel Learning Algorithms*

Some of the snapshot views of the backward data flow are shown in Fig. 6.

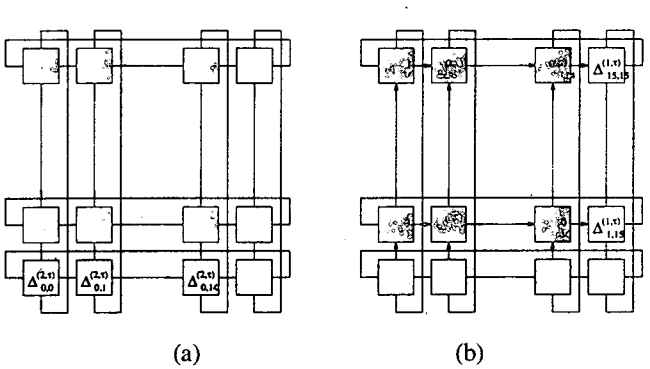


Fig. 6. Backward data flow for the learning phase of TDNN: (a) the error, (b) updating $\triangle$.

Before we proceed further, let us first define the notations: for each $PE_{i,j}$ we define $\triangle W_{i,j}^{(l,k)}, \Phi_i^{(l)}, \triangle_{(i,j)}^{(l,t)}$. In the learning process, the first phase occurs in $PE_{0,j}$:

$$\begin{cases} \triangle_{0,j}^{(2,t)} = (O_j - R_j) X_{0,j}^{(2,t)} f'(X_{0,j}^{(2,t)}), \\ \delta_{i,t}^{(2)} = \triangle_{o,j}^{(2,t)}, (j\in[0, N-1], t\in[T_2-1]. \end{cases} \quad (12)$$

$\triangle$ flows upwards and X in the right direction in the array to compute

$$\begin{cases} \triangle W_{i,j}^{(2,k)} = \alpha \frac{1}{T_2} \sum_{t=0}^{T_2-1} \triangle_{0,i}^{(2,t)} X_{j,15}^{(1,k+t)} + \eta \triangle W_{i,j}^{(2,k)}, \\ \triangle \Phi_i^{(2)} = \alpha \frac{1}{T_2} \sum_{t=0}^{T_2-1} \triangle_{0,i}^{(1,t)} + \eta \triangle \Phi_i^{(2)}, \\ (i\in[1, N], j\in[0, T_1-1], k\in[0, W_1-1]). \end{cases} \quad (13)$$

Here η and α are constants $\triangle W_{(i,j)}^{(2,k)}$ and $\triangle \phi_i^{(2)}$ are the previous incremental values, respectively. For the next lower layer, the delta becomes

$$\begin{cases} \triangle_{i,j}^{(1,t)} = \triangle_{i,j-1}^{(1,t)} + W_{j,i}^{(1,t)} \triangle_{0,j}^{(2,t)} \\ \triangle_{i,15}^{(1,t)} = f'(\triangle_{i,14}^{(1,t)}), (i\in[1, F_1], j\in[0, N-1], t\in[0, T_1-1]). \end{cases} \quad (14)$$

Using this delta, we can compute the weights:

$$\begin{cases} \triangle W_{i,j}^{(1,k)} = \alpha \frac{1}{T_1} \sum_{t=0}^{T_1-1} \triangle_{i,15}^{(1,t)} X_{0,j}^{(0,k+t)} + \eta \triangle W_{i,j}^{(1,k)}, \\ \triangle \Phi_i^{(1)} = \alpha \frac{1}{T_1} \sum_{t=0}^{T_1-1} \triangle_{0,i}^{(1,t)} + \eta \triangle \Phi_i^{(1)}, \\ (i\in[1, F_1], j\in[0, F_0-1], k\in[0, W_0-1]). \end{cases} \quad (15)$$

3. Overall Description

Putting the results together, one gets the overall program consisting of the three parts: initialization,recognition, and learning algorithms. The initialization algorithm consists of the three steps:

Initialization Algorithm:

1. Download programs into $\{PE_{i,j} \mid i \in [0, 15], j \in [0, 15]\}$.
2. Download η, and α into $\{PE_{i,j} \mid i \in [1; 15], j \in [0, 14]\}$.
3. Download look-up-tables for $f'(\cdot)$ into $\{PE_{0,j} \mid j \in [0, 14]\}$ and $\{PE_{i,0} \mid i \in [1, 15]\}$.

It is up to the host computer that downloads the programs, parameters, and look-up-tables into the EMIND-II hardware. Consequently, the recognition algorithm is summarized as

Recognition Algorithm:

1. Download: $X_{0,i}^{(0,t)} = x_{i,t}^{(0)}$, for $t \in [0, T_0-1]$, $i \in [0, T_0-1]$.
2. 1st layer:

$$\begin{cases} X_{i,j}^{(1,t)} = X_{i,j-1}^{(1,t)} + \sum_{k=0}^{W_1-1} w_{i,j,k}^{(1)} X_{0,j}^{(0,k+t)}, \\ X_{i,15}^{(1,t)} = f(X_{i,14}^{(1,t)}), \text{ for } i \in [1, F_1], j \in [0, F_0-1], t \in [0, T_1-1]. \end{cases}$$

3. 2nd layer:

$$\begin{cases} X_{i,j}^{(2,t)} = X_{i-1,j}^{(2,t)} + \sum_{k=0}^{W_2-1} w_{j,i,k}^{(2)} X_{i,15}^{(1,k+t)}, \\ X_{0,j}^{(2,t)} = f(X_{15,j}^{(2,t)}), \text{ for } i \in [1, F_1], j \in [0, N-1], t \in [0, T_2-1]. \end{cases}$$

4. Output: $X_{0,j} = \sum_{t=0}^{T_2-1} X_{0,j}^{(2,t)}$, for $j \in [0, N-1]$.
5. Store: $o_j = X_{0,j}$, for $j \in [0, N-1]$.
6. Go to Step 1.

Similarly one gets the learning algorithm:

Learning Algorithm:

1. Compute Steps 1-4 of the recognition algorithm.
2. Update delta:

$$\Delta_{0,j}^{(2,t)} = (O_j - R_j) X_{0,j}^{(2,t)} f'(X_{0,j}^{(2,t)}), \text{ for } j \in [0, N-1], t \in [T_2-1].$$

3. Update weights:

$$\begin{cases} \Delta W_{i,j}^{(2,k)} = \alpha \frac{1}{T_2} \sum_{t=0}^{T_2-1} \Delta_{0,i}^{(2,t)} X_{j,15}^{(1,k+t)} + \eta \Delta W_{i,j}^{(2,k)}, \\ \Delta \phi_i^{(2)} = \alpha \frac{1}{T_2} \sum_{t=0}^{T_2-1} \Delta_{0,i}^{(1,t)} + \eta \Delta \phi_i^{(2)}, \\ (i \in [1, M], j \in [0, T_1-1], k \in [0, W_1-1]). \end{cases}$$

4. Update delta:

$$\begin{cases} \Delta_{i,j}^{(1,t)} = \Delta_{i,j-1}^{(1,t)} + W_{j,i}^{(1,t)} \Delta_{0,j}^{(2,t)} \\ \Delta_{i,15}^{(1,t)} = f'(\Delta_{i,14}^{(1,t)}), (i \in [1, F_1], j \in [0, N-1], t \in [0, T_1-1]). \end{cases}$$

5. Update weights:

$$\begin{cases} \Delta W_{i,j}^{(1,k)} = \alpha \frac{1}{T_1} \sum_{t=0}^{T_1-1} \Delta_{i,15}^{(1,t)} X_{0,j}^{(0,k+t)} + \eta \Delta W_{i,j}^{(1,k)}, \\ \Delta \phi_i^{(1)} = \alpha \frac{1}{T_1} \sum_{t=0}^{T_1-1} \Delta_{0,i}^{(1,t)} + \eta \Delta \phi_i^{(1)}, \\ (i \in [1, F_1], j \in [0, F_0-1], k \in [0, W_0-1]). \end{cases}$$

6. Go to Step 1.

Whereas the recognition phase computes only the forward pass, the learning phase consists of the forward pass as well as the backward pass.

V. Discussion

Having derived how to implement TDNN, we are ready to analyze the algorithms. As an application,a word recognition is introduced. Finally, the parallel algorithms are compared with the serial algorithm in relation with computational complexity.

1. Fabrication

Figures 7 and 8 respectively are the photographs of the neurocomputer board and the die.

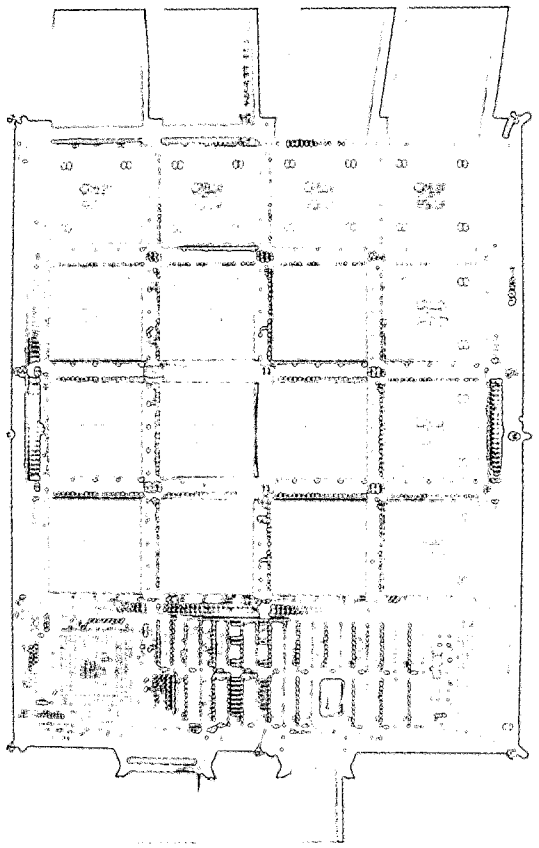


Fig. 7. Photograph of the EMIND_II neurocomputer.

As can be seen in the figure, the board consists of 16 DNP-IIs together with some driving circuits. Also, the board is connected to the host via flat cables. Figure 8 is the photograph of the DNP-II die. It is fabricated by 0.8 μ CMOS standard cell technology. The design goal was 40 MHz but the hardware test shows that it can operate as fast as 50 MHz clock speed.

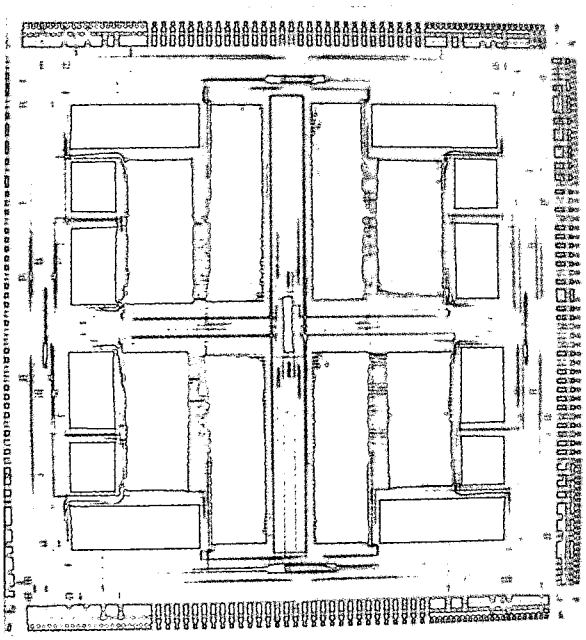


Fig. 8. Photograph of the DNP-II die.

2. Application to Word Recognition

As an application, we built a speech recognition system which consists of IBM PC486-DX2/66 and EMIND-II simulator. Fig. 9 provides the software structure of the word recognition system.

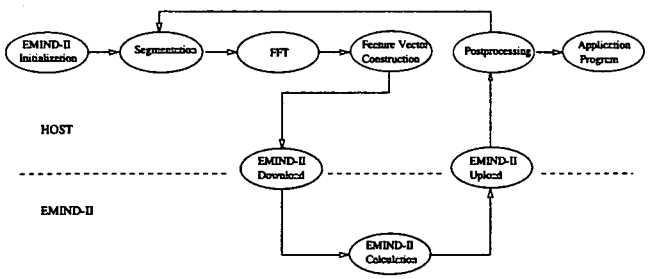


Fig. 9. Overall flow diagram.

At first the host must initialize the array by downloading parameters into the memory and input registers as the initialization algorithm describes. After that either the recognition algorithm or the learning algorithm must be executed. For the recognition phase, the input speech signal must undergo segmentation for end-point detection and thereafter feature vector construction by FFT transformation. This feature vector is loaded into the array, where estimated score of word class is determined. This result is collected in the computer by uploading and thereafter some post-processing must be done for natural language processing. The recognized sentence is utilized for further application such as robot arm manipulation in our case.

For the actual experiment, we have chosen 50 words for robotics control. From 30 men, we recorded 50 words for each man one time in common laboratory environment. Recording is done with 16 kHz sampling rate and 16 bit quantization. For these words, we prepared a database containing 2,100 learning patterns and 900 test patterns. The final results showed that the system displayed 97.10 % recognition rate for the learning patterns and 96.56 % for the test patterns. The result is only for test purpose, and comparison with others works has no meaning. Also although small vocabularies are used in our experiments, the network can be easily scaled up by modifying control parameters.

3. Computational Complexity

Let us now examine the computational complexity pertaining to the number of neurons to be emulated, weights to be stored, and multiplications as a sum of products. If we assume that the program size is P, then obviously the initialization takes $O(P)$ clocks. The look-up-table is avoided by executing a short program computing $f'(x) = f(x)(1 - f(x))$ for the sigmoid function.

Table 1. Serial algorithm vs. parallel algorithm.

Properties	Serial algorithm(total)	Parallel algorithm(per PE)
Number of neurons	$F_1T_1 + NT_2$ *(total)*	$\lceil \frac{F_1}{M(M-1)} \rceil T_1 + \lceil \frac{N}{M(M-1)} \rceil T_2$
Number of weights	$F_1(F_0W_0+1)+N(F_1W_1+1)$*(free weights)*	W_0+W_1
Sum of products for recognition	$W_0T_1F_0F_1+W_1T_2F_1N$	$W_0T_1\lceil \frac{F_0}{M-1} \rceil \lceil \frac{F_1}{M-1} \rceil + W_1T_2\lceil \frac{F_1}{M-1} \rceil \lceil \frac{N}{M-1} \rceil$
Sum of products for learning	$(W_0T_1F_0F_1+W_1T_2F_1N)+(T_2N+T_1F_1N)+(T_1T_2NF_1+T_0T_1F_1F_0)$	$(W_0T_1\lceil \frac{F_0}{M-1} \rceil \lceil \frac{F_1}{M-1} \rceil + W_1T_2\lceil \frac{F_1}{M-1} \rceil \lceil \frac{N}{M-1} \rceil + (T_2\lceil \frac{N}{M-1} \rceil + T_1\lceil \frac{F_1}{M-1} \rceil \lceil \frac{N}{M-1} \rceil) + (T_1T_2\lceil \frac{N}{M-1} \rceil \lceil \frac{F_1}{M-1} \rceil + T_0T_1\lceil \frac{F_1}{M-1} \rceil \lceil \frac{F_0}{M-1} \rceil$

Each row or column of the array plays a role of

$$\lceil \frac{F_1}{M-1} \rceil T_1 + \lceil \frac{N}{M-1} \rceil T_2 \qquad (16)$$

neurons. Here $\lceil x \rceil$ denotes the smallest integer not less than x. One can easily check that if M = 2, (16) is reduced to (2). Table I compares the performanceof serial and parallel algorithms in term of the number of neurons, weights, and sum of products.

In addition to other parameters, each neuron must store

$$W_0 + W_1 \tag{17}$$

weights.

The most time-consuming stage is the sum-of-products. Therefore, the recognition requires

$$W_0 T_1 \lceil \frac{F_0}{M-1} \rceil \lceil \frac{F_1}{M-1} \rceil + W_1 T_2 \lceil \frac{F_1}{M-1} \rceil \lceil \frac{N}{M-1} \rceil \tag{18}$$

multiplications. This relation also becomes (4) for M = 2.

Let us examine the learning phase. One pass of the learning consists of a forward pass and a backward pass as given by

This quantity consists of three terms: a forward pass, a backward delta update, and a backward weight update. Check that (7) is a special case of this equation with M = 2. This time must be multiplied by the number of passes and the number of patterns for each word. Notice that for the special case of M = 2, the equations (16), (18), and (19) become (2), (4), and (7), respectively. For a given network of size M, a PE acts as many virtual neurons and the network is M^2 times faster than the serial system.

Putting it altogether, one can obtain Table I. It is immediately apparent that the potential power of this system

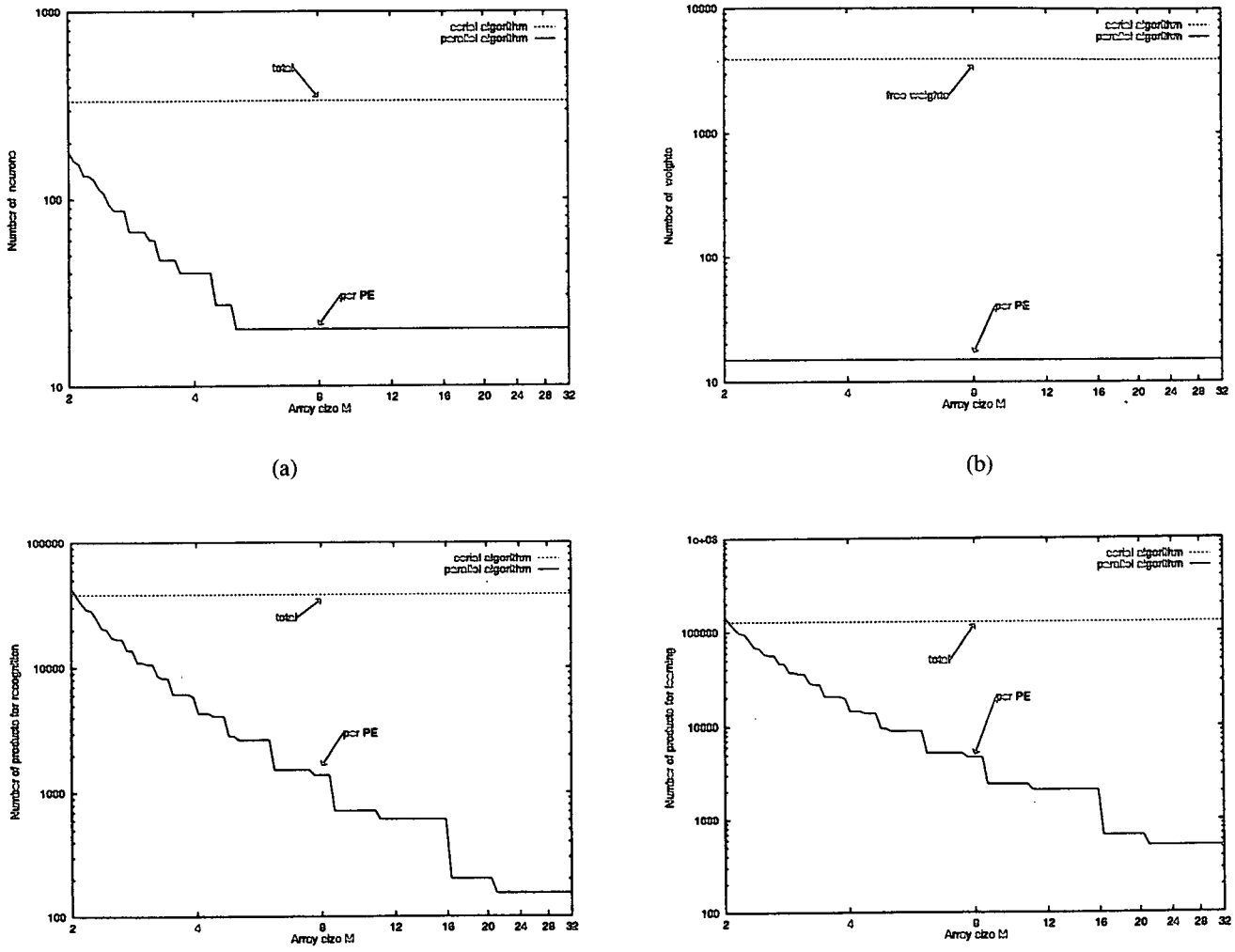


Fig. 10. Computational complexity vs. M the array size(F_0=15, T_0=20, W_0=8, F_1=15. W_1=7, and N=20): (a) number of neurons, (b) number of weights, (c) number of products for recognition, and (d) number of products for learning.

$$\begin{aligned}&(W_0 T_1 \lceil \frac{F_0}{M-1} \rceil \lceil \frac{F_1}{M-1} \rceil + W_1 T_2 \lceil \frac{F_1}{M-1} \rceil \lceil \frac{N}{M-1} \rceil) + \\ &(T_2 \lceil \frac{N}{M-1} \rceil + T_1 \lceil \frac{F_1}{M-1} \rceil \lceil \frac{N}{M-1} \rceil) + \\ &(T_1 T_2 \lceil \frac{N}{M-1} \rceil \lceil \frac{F_1}{M-1} \rceil + T_0 T_1 \lceil \frac{F_1}{M-1} \rceil \lceil \frac{F_0}{M-1} \rceil)\end{aligned} \tag{19}$$

resides in that each PE can be time-shared by many neurons. However, owing to the limited size of weight memory, the number of virtual neurons is quite limited. An approach to a large system is using some batch processing technique. In this case, it is not necessary to load all the weights in the

array. Instead, one partition of the weights that is utilized by the array is downloaded and processed one by one. This scheme requires a fast hardware of load and store operations.

Fig. 10 displays the computational complexity as a function of the array size parameter M. The other parameter values are F_0 = 15, T_0 = 20, W_0 = 8, F_1 = 15, W_1 = 7, and N = 20.

In this figure, the coordinates are drawn in log scale. Notice that the computational load for each PE is gradually decreased as the network size increases.

VI. Conclusion

This paper introduces a versatile neurochip DNP-II and the neurocomputer EMIND-II. Considering the particular nature of the wavefront toroidal mesh-array, we developed efficient algorithms for realizing TDNN by EMIND-II.

Based on this algorithm, a speech recognition system is built. Experimental results show that this speech recognition system built on neural network hardware works well showing advantages of fast computation speed and high recognition rate.

As a conclusion, the major achievement of this system is its on-chip learning capability, scalability for a large virtual machine, fast speed for real-time applications, digital interface, wavefront mesh-array structure suitable for parallel/pipelined computation, and the possibility of building a wide variety of neural networks.

References

[1] D. Hammerstrom, "A VLSI architecture for high-performance, low-cost, on-chip learning," *Proceedings of the International Joint Conference on Neural Networks*, pp. 537-544, 1990.

[2] A. Hiraiwa et al., "A two level pipeline RISC processor array for ANN," *Proceedings of the International Joint Conference on Neural Networks*, pp. 137-140, 1990.

[3] Kato et al., "A parallel neurocomputer architecture towards billion connection updates per second," *Proceedings of the International Joint Conference on Neural Networks*, pp. 47-50, 1990.

[4] J. R. Nicholls, "The design of the MasPar MP-1: A cost effective massively parallel computer," *proceedings of the COMPCON 90*, pp. 25-28, 1990.

[5] T. Nordstrom and B. Svenson, "Using and designing massively parallel computers for artificial neural networks," *Journal of Parallel and Distributed Computing*, vol. 14, pp. 260-285, 1992.

[6] D. Hammerstrom, "Neural networks at work," *IEEE Spectrum*, pp. 26-32, June 1993.

[7] M. W. Kim, Y. J. Lee, H. B. Lee, J. S. Lee, J. M. Kim, J. H. Kim, S. H. Oh, C. D. Lim, and H. K. Song, "E-MIND: An implementation of a digital neurocomputer and its application to handwritten digit recognition," *Proceedings of the International Joint Conference on Neural Networks*, pp. 258-263, 1992.

[8] M. W. Kim, Y. J. Lee, C. M. Kim, and Y. S. Song, "A wavefront array processing architecture for real-time simulation of large scale neural networks," *Proceedings of the International Joint Conference on Neural Networks*, pp. 1959-1962, 1993.

[9] M. W. Kim, J. M. Kim, Y. S. Song, Y. J. Lee, and H. B. Lee, "An asynchronous inter-processor communication based, input recycling parallel architecture," *Proceedings of the World Conference on Neural Networks*, pp. 576-583, 1994.

[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, MIT Press, 1986.

[11] K. J. Lang and A. H. Waibel, "A time-delay neural network architecture for isolated word recognition," *Neural Networks*, vol. 3, pp. 32-43, 1990.

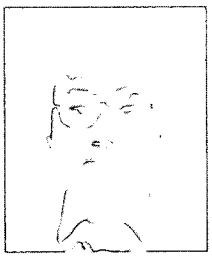
[12] J. J. Hopfield, "Neural network and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences*, pp. 2554-2558, 1982.

[13] T. Kohonen, "Self-organized formation of topologically correct feature map," *Biological Cybernetics*, vol. 43, pp. 59-69, 1982.

[14] U. Ramacher, "SYNAPSE-a neurocomputer that synthesizes neural algorithms on a parallel systolic engine," *Journal of Parallel and Distributed Computing*, vol. 14, pp. 306-318, 1992.

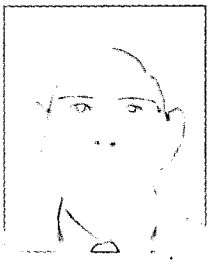
[15] J. Wawrzynek, K. Asanovic, and N. Morgan, "The design of a neuro-microprceessor," *IEEE Transactions on Neural Networks*, vol. 4, no. 3, pp. 394-399, 1993.

[16] J. B. Hampshire and B. Pearlmutter, "Equivalence proofs for multilayer perceptron classifiers and the Bayesian discriminant function," *Proceedings of the 1990 connectionist models*, pp. 159-172, 1991.



Hong Jeong was born in Seoul, Korea, in 1953. He received the B.S. degree in the Department of Electrical Engineering from Seoul National University in 1977. In 1979, he received the M.S. degree in the Department of Electrical Engineering from Korea Advanced Institute of Science and Technology. In 1984, 1986, and 1988, he received the S.M., E.E., and Ph. D. degrees, respectively, all in the Department of Electrical Enginnering and Computer Science at M.I.T., Cambridge, Massachusetts, U.S.A. During the period of 1979-1982, he was a faculty staff at the Department of Electrical Engineering at Electronics and Electrical Enginnering at the Pohang University of Science and Technology, where he now works as an Associate Professor. He is Sigma Xi member. During 1994-1995, he worked as a vice-chairman in the Special Interest Group on Neurocomputing in the Korea Information Science Society. Also from 1991, he has worked as a committee staff in the Neural Networks, Fuzzy and Artificial Intelligence Group in the Korean Institute of Telematics and Electronics. His research interests include neural networks and speech recognition.

Cha-Gyun Jeong was born in Jeon-Ju, Jeon-Buk, Korea, in 1966. He graduated from the Korea Advanced Institute of Science and Technology with B.S. degree in Electronics Engineering in 1990, and received an M.S. degree from Pohang Institute of Science and Technology with B.S. degree in Electronics Engineering in 1990, and received an M.S. degree from Pohang Institute of Science and Technology (POSTECH) in Electrical and Electronic Engineering in 1992, respectively. Since 1992, he has been working towards the Ph. D. degree in the Electrical Engineering from POSTECH. His research interests are speech recognition and neural networks.



Myung-Won Kim graduated from Seoul National University, Seoul Korea with B.S. in Applied Mathematics in 1972. He received an M.S. and Ph.D. in Computer Science from University of Massachusetts at Amgerst in 1981 and University of Texas at Austin in 1986, respectively. He worked with AT&T Bell Laboratories from 1985 to 1987 and he also supervised a neural network research group at Electronics and Telecommunications Research Institute in Korea from 1987 to 1994. He is currently an associate professor in School of Computing, Soongsil University, Korea. His research interests include neural networks, pattern recognition, knowledge representation, machine learning, and parallel implementation of neural networks.