Neural Network Architectures for Robotic Applications

SUN-YUAN KUNG, FELLOW, IEEE, AND JENQ-NENG HWANG, MEMBER, IEEE

Abstract—**This paper proposes a ring VLSI systolic architecture for implementing artificial neural networks (ANN's) with applications to robotic processing. Key design issues on algorithms, applications, and architectures are examined. A variety of neural networks are considered, including single-layer feedback neural networks, competitive learning networks, and multilayer feed-forward networks. It is demonstrated that the ANN's are suitable to all three levels of robotic processing applications, including task planning, path planning, and path control levels. For these applications, a programmable systolic array is developed, which can exploit the strength of VLSI to provide intensive and pipelined computing. Both the retrieving and learning phases are integrated in the design. The proposed architecture is more versatile than other existing ANN's; therefore, it can accommodate all the useful neural networks for robotic processing.**

I. Introduction

THE POWER of neural networks hinges upon their distinct features of robust processing and adaptive capability in changing and noisy environments. It is estimated that the human brain contains over 100 billion (10^{11}) neurons. Neurons posses tree-like structures (called *dendrites*) specialized to receive incoming signals from other neurons across junctions called *synapses*. From each neuron, there is a single output fiber (called an *axon*) specialized to propagate action potentials so that the activation values of each neuron can be transmitted to many other neurons. Studies of brain neuroanatomy indicate more than 1000 synapses on the input and output of each neuron. In other words, although the neurons' transition time of a few *milliseconds* is about a million-fold times slow than current computer elements, the brain has a thousand-fold greater connectivity than today's supercomputers [16].

It has been postulated that the primary function of neocortical networks in the cerebrum is to form internal representations of classes and subclasses of objects, using perceptron-like algorithms. Moreover, it has been observed that the cerebellum functions as an associative content addressable memory (ACAM) that can be "trained" by the cerebrum. Many examples of biological applications are also available as a useful clue for the development of artificial neural networks (ANN's). In the example of natural vision processing, edge information is extracted in part in the retina via lateral inhibition between retinal neurons. In the cortex, there is a lateral excitation process which computes the brightness of a patch of an image bounded by edges. On the other hand, in primates, depth perception is formed by comparing images from the two eyes. Finding the correct overall assignment (from the several depth assignments existing in each cortical neighborhood) takes numerous trials before the cortical network finds a "solution." Interestingly, this process is analogous to the *relaxation* process used in many numerical algorithms run on current digital computers [6], [15].

The state of the art of the ANN's has embraced a very broad scope of neural processing models, from the earlier experimental research work on digital logic networks [51] in the 1940's, perceptrons [63], Adalines [78], and learning matrix [69] in the 1960's, to the correlation matrix linear models of ACAM networks [5], [7], [35], [79], and the cooperative-competitive neural network models [18], [21] in the 1970's. Extending earlier works by Steinbush [69], Willshaw [79], Amari [5], Cohen [14], Hopfield proposed a very popular iterative computational model for associative retrieval and optimization [25]–[27]. In addition to some significant contributions to the memory and learning models using competitive learning for autonomous feature extraction [66] and delta learning for generalized information storage [50], Rumelhart and his colleagues revived the back-propagation (generalized delta) learning algorithm for the multilayer perceptrons [53], [55], [64], [76], which has been successfully used in many experimental works [11], [20], [43], [44], [67].

There are three important aspects of ANN research can be identified: *algorithms, applications*, and *architectures*.

A. Algorithmic Aspects of ANN's

A basic ANN model consists of a large number of neurons, linked to each other with connection weights (see Fig. 1). Each, say ith, neural processing unit (PU) has an activation value a_i. This value (either discrete or continuous) is propagated through a network of unidirectional connections to other PU's in the network. Associated with each connection, there is a *synaptic weight* denoted as w_{ij} which indicates the effect the jth PU has on the ith PU. From algorithmic point of view, the ANN processing can be divided into two phases: *retrieving* and *learning*.

Retrieving Phase: Suppose that the connectivity pattern and weights of a neural network is known, in response to inputs

Manuscript received December 16, 1987; revised July 5, 1988 and May 1, 1989. This research was supported in part by the National Science Foundation under Grant MIP-87-14689 and by the Innovative Science and Technology Office of the Strategic Defence Initiative Organization, administered by the Office of Naval Research under contract N00014-88-K-0515

S. Y. Kung is with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544

J. N. Hwang was with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544. He is now with the Department of Electrical Engineering, University of Washington, Seattle, WA 98195

IEEE Log Number 8929717

1042-296X/89-1000-0641$01.00

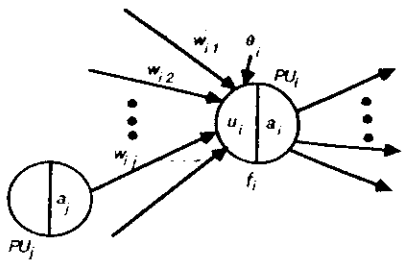


Fig. 1. A basic ANN model with two operations: propagation rule, and nonlinear activation.

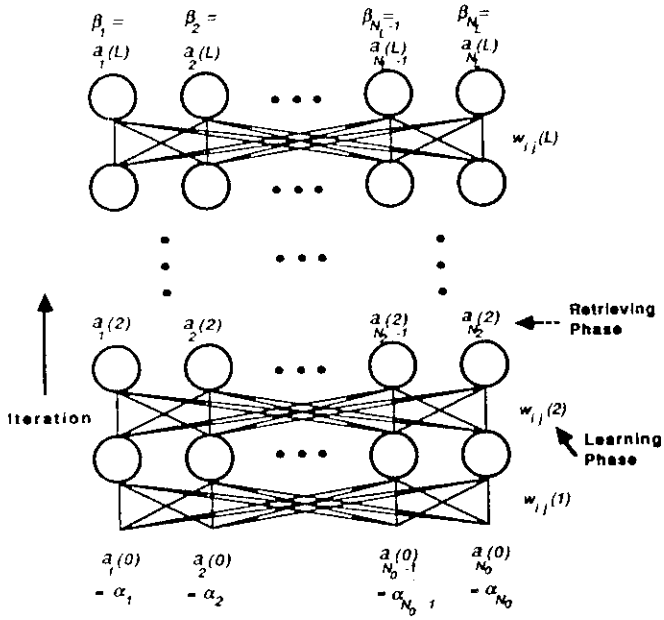


Fig. 2. A generic iterative model (L-iterations) for ANN's, where $w_{ij}(l)$ and N_l may be homogeneous or heterogeneous with respect to l.

(test patterns), the retrieving phase performs the iterative updating of activation values of each neuron based on the *system dynamics* to produce the responding outputs. The system dynamics in the retrieving phase of an ANN model can be written as a generic iterative formulation [41]

$$u_i(l+1) = \sum_{j=1}^{N_l} w_{ij}(l+1)a_j(l) \tag{1}$$

$$a_i(l+1) = f_i(u_i(l+1),\ \theta_i(l+1)) \tag{2}$$

where $1 \le i \le N_{l+1}$ and $0 \le l \le L-1$. The iteration index l can represent either *time* or *spatial* iterations.

Two types of inputs can be used to represent the test/training patterns: the stimulus inputs $\{a_i(0)\}$ and the external inputs $\{\theta_i(l)\}$. The initialization activation values are often denoted by α_i, i.e., $\{a_i(0) = \alpha_i\}$. The termination activation values are often denoted by β_i, i.e., $\{a_i(L) = \beta_i\}$.

The system dynamics in (1) and (2) may be graphically represented by an L-level feed-forward neural network (with N_l neural units at lth level) shown in Fig. 2, where one mathematical iteration is corresponding to one level of the network. Each PU, say ith neuron at $(l+1)$th iteration, receives the weighted inputs from other PU's at lth iteration to yield the net input $u_i(l+1)$ according to the *propagation rule* [see (1)]. The net input value $u_i(l+1)$, along with the external input $\theta_i(l+1)$, will determine the new activation value $a_i(l+1)$ by the *nonlinear activation* function f_i [see (2)]. The nonlinear activation functon f_i can be a deterministic function, winner-take-all mechanism [44], or a stochastic decision [1], [60] (for simpler notation, we will denote the position-invariant activation function as f). In some classification applicatons, winner-take-all type of nonlinear mechanism is adopted, which can be easily implemented by lateral inhibitions so that only the neuron receiving largest input is activated.

Learning Phase: Based on the input and/or target training patterns, the learning phase performs the iterative updating of the synaptic weights for all the connections based on the adopted *learning rule*. The weight updating (credit assignment) problem is to find a set of synaptic weights so as to optimize certain predefined measure function E based on a set of training patterns. The learning phase usually involves two steps: In the first step, the input training patterns are processed by the network based on the retrieving phase equations and generate some actual responses. In the second step, the weights are updated according to the responses generated and the chosen learning rules. Often recursive procedures are adopted. A common recursive weight updating formulation (learning rule) for an ANN model can be given [41] by

$$w_{ij}(l) \Leftarrow w_{ij}(l) + \eta \Delta w_{ij}(l). \tag{3}$$

The new weight value can be determined by the current weight value, the updating rate parameter η, and most importantly the increment of weight change. The updating rate η is introduced to regulate the rate of change of each weight at each recursion, it can be a global constant or can be a locally dependent variable $\eta_{ij}(l)$. For a simpler notation, we will use η to represent $\eta_{ij}(l)$ in the following discussions.

The learning recursion can represent either type of the two possible recursions: *pattern* and *sweep*. In the case of pattern recursion, the network updates the synaptic weights after the presentation of each training pattern. On the other hand, in the case of sweep recursion, the network updates the synaptic weights after the presentation of all the training patterns.

B. Applicational Aspects of ANN's

Neural neworks have been successfully applied to many applications which may be grouped into two classes: *optimization* and *associative retrieval/classification*.

Optimization: For the optimization application, the neural networks are used as a state-space search mechanism. The (fixed) synaptic weights are set before the search process is started. There is no explicit learning procedure for setting the weights and the dervation of $\{w_{ij}\}$ usually hinges upon finding a Liapunov energy function for the specific application.[1] For example, if the Hopfield neural network is used for solving an optimization problem, the Liapunov energy function has the

[1] A Liapunov energy function is monotonically decreasing along time evolution, i.e., $(d/dt)\,E \le 0$.

following form [26], [27]:

$$E = -\frac{1}{2}\sum_i \sum_j w_{ij} a_i a_j - \sum_i \theta_i a_i. \quad (4)$$

In some applications, the Liapunov energy function is directly available (e.g., regularized least squares applications [38]). In others, the Liapunov energy function has to be derived from a given cost function and the constraints in the optimization problem (e.g., traveling salesman problem [27]). Once the synaptic weights are determined from the energy function, then the retrieving phase can be performed by following the system dynamics in (1) and (2). Basically, the iterations execute a gradient descent of the energy function until they converge to a (local) optimal state.

In order to reach the global optimal solution, one can resort to the Boltzmann machine, which adopts the same energy function as the Hopfield network and the simulated annealing technique [24]. Simulated annealing is a search technique that allows the possibility of getting out of the trap of the local optimum by introducing a mechanism of flattening the trap and a possibility based on a stochastic decision of accepting an updating which (temporally) corresponds to a *worse* solution [33]. It is claimed that, with a proper annealing schedule, the state of the Boltzmann machine will gradually move toward the global optimal solution.

Associative Retrieval/Classification: Another very promising application of neural processing is for associative retrieval or associative classification. The associative retrieval is to retrieve the complete pattern, given partial information of the desired pattern (auto-association), or to retrieve a corresponding pattern in subset B, given a pattern in subset A (pattern association). The associative classification is to identify the corresponding category for any test pattern. The retrieving phase uses the same system dynamics equations (1) and (2) as in the optimization applications. For this application class, some learning schemes are often adopted to train the synaptic weights.

Robotic Applications: Most robotic problems currently being attacked can be categorized into either of the three processing levels: task planning (e.g., depth determination and arm–camera coordination), path planning (e.g., robot navigation), and path control (e.g., motor control). Most of these robotic processing can be formulated in terms of optimization or pattern association problems. Therefore, neural networks can naturally be adopted to solve the robotic processing tasks. For example, stereo vision for task planning [40], [70], [71], [80], autonomous robot path planning [31], [74], manipulator position control [46], [75], can be formulated as optimization problems. On the other hand cerebellar model articulation controller (CMAC) [3], [48], [52], sensor/motor control [42], [62], global terrain retrieval [31], and voluntary movement control [32], [58] can be formulated as pattern association tasks. For more detailed discussion of using neural networks for robotic applications, the reader is referred to Section III.

C. Architectural Aspects of ANN's

Neural networks have been proposed to perform computations "in the style of the brain," relying on massive parallelism and massive interconnectivity of their simple processing units ("neurons"). Recently, a discussion on the possible ways to implement neural network models have been conducted in the research community with a major division into "analog" and "digital" cams. In our view, both analog and digital implementations can be useful when used in areas where they are strongest. Analog neural networks must be used in the pre-processing stages for data compression and initial interpretation since the data rates in real-world applications usually exceed the capabilities of digital networks (e.g., vision). On the other hand, preprocessing stages usually require less accuracy thus being suitable for analog networks. In later processing stages, assuming an initial data reduction and interpretation has been performed, the flexibiliy requirements call for the digital implementation where we can take advantage of their capability of partitioning large problems and implementing different networks with the same hardware.

Analog neural network implementations offer great speed but little else since they suffer in terms of accuracy, density, learning capability, flexibility, and do not scale well with the size of problem since scaling introduces additional noise in the device thus limiting the accuracy even more [46]. The main drawback of analog neural networks however is in their inability to solve problems that require a number of neurons greater than the network's physical size since there is no natural way to store intermediate values, reprogram the network, and then combine the results of multiple passes. A related drawback is the difficulty of implementing hierarchies of interconnected networks that cooperate to solve a large problem.

Digital implementations of neural networks, on the other hand, offer a mature and well understood technology, great accuracy, and scale much better than analog implementations. Using digital logic and memory, it is quite easy to partition a large problem so that it can be solved by a smaller (in terms of hardware) implementation. Using the same logic and memory, a digital implementation can realize more than one network and combine the results in hierarchical fashion to solve large problems. The main drawbacks of digital VLSI implementations are their larger silicon area, relatively slower speed, and the great cost of interconnecting processing units. Summarized below are several key considerations for parallel processing and array architecture implementations of neural networks:

- Convergence issues of synchronous (parallel) updating of system dynamics in the retrieving phase.
- The architectural design should ensure that the processing in both the retrieving phase and the learning phase share the same array configuration, storage, and processing hardware. This will not only speed up real-time learning but also avoid the difficulty in the reloading of synaptic weights for retrieval.
- The digital design must identify a proper digital arithmetic technique to efficienctly compute the necessary operations required in both phases.
- The array architecture design should maximize the strength of VLSI in terms of intensive and pipelined computing and yet circumvent its main limitation on communication. It is desirable to find a local interconnectivity of systolic

solution to the implementation of the global interconnectivity of neural networks.

- A digital design must offer a greater flexibility, so a general-purpose programmable array architecture is preferred for implementing a wide variety of neural network algorithms in both retrieving and the learning phases.

This paper is organized as follows: In section II we discuss several neural networks models useful for robotic processing. Section III shows how to apply the neural networks models for all levels of robotic processing. Finally, in Section IV, we propose a universal ring systolic architecture for implementing all the useful neural networks for robotic processing. Implementation considerations of neural processing units are also addressed.

II. Useful Neural Networks for Robotic Processing

Single-layer feedback neural networks have been shown to serve as a state-space search mechanism, which can be useful for pattern association and optimization tasks. Competitive learning networks provide a very efficient unsupervised learning mechanism for a lot of pattern classification applications. Due to the capability of establishing internal representation by the hidden neurons, the feed-forward multilayer perceptrons have demonstrated very powerful retrieving/learning capabilities.

A. Single-Layer Feedback Neural Networks

There are many single-layer neural networks. Some of them are feedback networks, e.g., the Hopfield associative/optimization network [25], [28], the Boltzmann machine [1], and the Rumelhart's memory/learning module [50].

1) Hopfield Associative/Optimization Neural Networks: Hopfield neural networks have found many applications, including pattern recognition [17], vowel classification [8], combinatorial optimization problem [27], [72], linear programming problem [73], computational vision [34], and parameter estimation [61].

A Hopfield associative neural network is a single-layer (time-iterative) feedback network, which consists of N binary-valued neurons linked to each other with symmetric weights $\{w_{ij} = w_{ji}\}$. In the Hopfield model, thresholding elements are added to linear associators to perform iterative feedback auto-association tasks. Moreover, a notion of energy function is adopted to prove that the feedback system exhibits a number of locally stable points (attractors) in the state space, which provides a basic mechanism for signal retrieval and error correction from partial or noisy missing information [25].

Retrieving Phase: The system dynamics in the retrieving phase of a Hopfield associative network is

$$u_i(l+1) = \sum_{j=1}^{N} w_{ij} a_j(l)$$

$$a_i(l+1) = \begin{cases} 1, & \text{if } u_i(l+1) > -\theta_i \\ 0, & \text{if } u_i(l+1) < -\theta_i \\ a_i(l), & \text{if } u_i(l+1) = -\theta_i. \end{cases} \tag{5}$$

The original Hopfield associative network requires each neuron to be updated asynchronously (sequentially) to guarantee the convergence of the system dynamics. The dynamic evolution of the system state can be regarded as an energy minimization that continues until a stable state (local energy minimum) is reached. To demonstrate the convergence, a notion of Liapunov energy function is often instrumental.

In the following, we will show that synchronous (parallel) updating of all the neurons at each iteration is also possible for non-negative definite symmetric weight matrix $\{w_{ij}\}$. To demonstrate the convergence of the parallel updating iterations, it is useful to adopt the following Liapunov energy function $E(l)$ (after the lth updating)

$$E(l) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} a_i(l) a_j(l) - \sum_{i=1}^{N} \theta_i a_i(l).$$

If all the neurons are updated in parallel at each time iteration, then the energy change between any two iterations is

$$\begin{aligned}
\Delta E(l+1) &= E(l+1) - E(l) \\
&= -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} a_i(l+1) a_j(l+1) - \sum_{i=1}^{N} \theta_i a_i(l+1) \\
&\quad + \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} a_i(l) a_j(l) + \sum_{i=1}^{N} \theta_i a_i(l) \\
&\quad - \frac{1}{2}\sum_{i=1}^{N} (a_i(l+1) - a_i(l)) \left[\sum_{j=1}^{N} w_{ij}(a_j(l+1) - a_j(l)\right] \\
&\quad - \frac{1}{2}\sum_{i=1}^{N} (a_i(l+1) - a_i(l)) \left[\sum_{j=1}^{N} w_{ij}(a_j(l+1) - a_j(l)\right] \\
&= -\Delta \boldsymbol{a}^T(l+1)[\boldsymbol{u}(l+1) + \theta] \\
&\quad - \frac{1}{2}\Delta \boldsymbol{a}^T(l+1) \boldsymbol{W} \Delta \boldsymbol{a}(l+1) \\
&= \Delta E_1(l+1) + \Delta E_2(l+1).
\end{aligned} \tag{6}$$

Since the nondecreasing thresholding functions are assumed, the signs of $\{\Delta a_i(l+1)\}$ and $\{u_i(l+1) + \theta_i\}$ are the same (see (5)), and $\Delta E_1(l+1) \le 0$. In order to also guarantee $\Delta E_2(l+1) \le 0$, the weight matrix should be a non-negative definite matrix, this can be ensured by setting weight matrix to be

$$w_{ij} = \sum_{p=1}^{P} (2\beta_i^{(p)} - 1)(2\beta_j^{(p)} - 1)$$

or

$$\boldsymbol{W} = \sum_{p=1}^{P} [2\boldsymbol{v}^{(p)} - \boldsymbol{1}][2\boldsymbol{v}^{(p)} - 1]^T \tag{7}$$

where P binary reference patterns, represented by $\{\boldsymbol{v}^{(p)} = [\beta_1^{(p)}, \beta_2^{(p)}, \cdots, \beta_N^{(p)}], p = 1, 2, \cdots, P\}$, are stored in and to be retrieved from the associative network. Note that the $\boldsymbol{W}$ matrix is formed by an outer product without diagonal

nullification (as opposed to original Hopfield network [25]), so it is a non-negative definite matrix.

Learning Phase: The weight determination process based on (7) can be interpreted by the recursive weight updating formulation given in (3), i.e.,

$$w_{ij} \leftarrow w_{ij} + \frac{\eta}{2}\beta_i\beta_j \tag{8}$$

where the recursion index is corresponding to each training pattern used. This leads to the simplest form of Hebbian learning rule [22], and the stored pattern $\{\beta_i\}$ is the desired output (desired auto-associative retrieval information).

Instead of going through the iterative Hebbian weight updating learning procedures, Hopfield used the ensemble average of $\{w_{ij}\}$ over P (training) pattern recursions to determine the fixed connection weights with approximate *zero statistical mean* [25], this leads to the weight determination formulation given in (7).

Hopfield Optimization Neural Networks: In order to imitate the continuous input-output relationship of real neurons and to simulate the integrative time delay due to the capacitance of real neurons [26], Hopfield modified the associative network by adopting the following *sigmoid activation* function $f(x)$:

$$f(x) = \frac{1}{1 + e^{-x/u_0}} \tag{9}$$

and proposed the following dynamic system for the retieving phase [26], [72]:

$$u_i(l+1) = \sum_{j=1}^{N} w_{ij}a_j(l) \tag{10}$$

$$a_i(l+1) = \frac{1}{1 + e^{-u_i'(l+1)/u_0}} \tag{11}$$

where $u_i'(l+1) = \kappa_1(u_i(l) + \theta_i) + \kappa_2(u_i(l+1) + \theta_i)$, and κ_1 and κ_2 are proper constants proposed in [72]. Note that when the coefficient u_0 tends to zero, then the activation function of the Hopfield neural network becomes a *step function* as given in (5).

In the continuous state, the convergence with parallel updating can be assured only when the (time step) coefficient (κ_2) is small enough [56]. This will lead us to the small values $\{\Delta \boldsymbol{a}^T(k+1)\}$, and the contribution of the energy term $\Delta_k E_2$ in (6) can be neglected, so that $\Delta_k E_1$ dominates the level of the energy change.

2) Boltzmann Machine: If we substitute the sigmoid coefficient u_0 in the Hopfield neural networks (see (11)) by a temperature-control parameter T, then (11) becomes

$$a_i(l+1) = f(u_i(l+1), \theta_i) = \frac{1}{1 + e^{-(u_i(l+1)+\theta_i)/T}} \tag{12}$$

where we let $\kappa_1 = 0$ and $\kappa_2 = 1$ (i.e., no tine delay assumed).

Retrieving Phase: Let us confine the neuron response to be discrete value (0 or 1), and let the deterministic activation operation in the Hopfield neural network be replaced by a stochastic process defined below

$$\Pr\,(a_i(l+1) = 1) = \frac{1}{1 + e^{-(u_i(l+1)+\theta_i)/T}}. \tag{13}$$

More precisely, $a_i(l+1)$ is set to "1" with probability given in (13), otherwise $a_i(l+1) = 0$. It was shown in [24] that, the system dynamics given in (13) ensures that "in thermal equilibrium the relative probability of the two global states is determined solely by their energy difference, and follows a Boltzmann distribution." Therefore, this modified version of the Hopfield neural network is called the *Boltzmann machine* [1], [24]. If T follows an *annealing schedule*, i.e., T gradually decreases during the iterations of neural networks, the sigmoid activation operation in (11) becomes an *annealing* operation. With a proper annealing schedule, the states of the Boltzmann machine will gradually move toward the global optimal solution.

Weight Determination: The weight determination procedure, basically the same as that for the Hopfield network, starts with finding the energy function. The synaptic weights can be determined by comparing this energy function with (4). The retrieving phase can be performed by the system dynamics in (13).

Learning Phase: While the retrieving phase of a Boltzmann machine is similar to the Hopfield net, the Boltzmann learning rule has a very different form. It may be regarded as a two-step batch-updating Hebbian rule [1], [4], [65]: In the first step (denoted *phase*$^+$), some outputs of the neural units in the network are clamped to predetermined desired values, and then let the neural net iterates through a proper annealing schedule based on (13) until convergence. In the second step (denoted *phase*$^-$), none of the neural units are clamped and the network again iterates (following the same annealing schedule) until it reaches a new convergent state. If there exists any discrepancy between the convergent states for the *phase*$^+$ and the *phase*$^-$, then the synaptic weights should be adjusted as follows:

$$\Delta w_{ij} = \eta(p_{ij}^+ - p_{ij}^-) \tag{14}$$

where p_{ij}^+ (respectively, p_{ij}^-) represents the correlation of the two neurons i and j in the *phase*$^+$ (respectively, in the *phase*$^-$). They can be determined by the average probability that both neurons are on (i.e., $\beta_i = \beta_j = 1$) over many training patterns. Instead of modifying the weights immediately after the exposure of each training pattern as in the conventional Hebbian learning rule (see (8)), the weight adjusting (i.e., (14)) is performed only after enough training patterns are taken. The goal of this learning rule is to find a set of synaptic weights such that the activation outputs in the *phase*$^-$ match the desired outputs in the *phase*$^+$ as closely as possible.

B. Competitive Learning Networks

Competitive learning is an unsupervised (self-organized) procedure that classify a set of input patterns into a number of

disjoint clusters in such a way that the input patterns within each cluster are all similar to one another [23]. Most competitive learning networks are single-layer feed-forward networks using winner-take-all mechanism. It is possible to cascade several individually optimized single-layer competitive learning networks to establish a hierarchical classification network [19], [66].

Retrieving Phase: Without loss of generality, the system dynamics between the inputs $\{\alpha_i\}$ and the outputs $\{\beta_i\}$ in the retrieving phase of a competitive learning network is given

$$u_i = \sum_{j=1}^{N_0} w_{ij}\alpha_j$$

$$\beta_i = \begin{cases} 1, & \text{if } u_i > u_k,\ \forall k \\ 0, & \text{if else} \end{cases} \tag{15}$$

where $1 \le i \le N_1$. The *winner-take-all* nonlinear competition mechanism (e.g., lateral inhibitions) is used in the output layer so that only the neuron receiving largest input is activated.

Learning Phase: The learning phase in the competitive learning network can also be interpreted by the recursive weight updating formulation (see (3))

$$w_{ij} \Leftarrow w_{ij} + \eta\beta_i(\alpha_j - w_{ij}) \tag{16}$$

where one recursion is for one pattern. According to (15), (16) implies that only the weights associated with the winning neuron are updated and all the other weights remain unchanged. This is a special feature of the competitive learning networks.

We hasten to note that the above formulation can only describe the basic feature commonly shared by the competitive learning networks. In actuality, each individual model has almost unexceptionally adopted certain special mechanism.

- Kohonen's self-organized feature map [36], [37] introduced a neighborhood (whose size slowly decreases with each iteration) of a winning neuron in a two-dimensional neuron layer. Weights associated with the winner and the neurons in the neighborhood of the winner are all modified. This has purpose of making the neurons more responsive to the current input pattern.
- The adaptive resonance theory (ART) by Carpenter and Grossberg [12], [21] introduced a *vigilance test* to adaptively create new neuron units for the incoming input patterns which are quite different from the memorized patterns. This test requires additional modifiable feedback weights connecting from output layer to the input layer.
- Rumelhart's competitive learning algorithm [66] introduced a *leaky learning model* to prevent the possibility of totally unlearned neurons. This is done by performing training in (16) over all the weights in the network. However, the weights associated with the winner get much larger η values.
- Neocognitron can be formed as a hierarchical learning system by cascading many single-layer competitive learning networks [19]. The learning for each layer is largely based on single-layer analysis. It progresses stage from the input layer to the output layer.
- Darwin networks were created to explore the brain theory called neural Darwinism. These networks select those neuronal groups (called repertoire) which respond best to the specific input stimuli [62]. Neuron groups in a repertoire respond best to overlapping but similar input patterns due to the randomness of neuronal growth, the response to important unexpected inputs is thus insured [46].

C. Multilayer Perceptrons

Multilayer feed-forward neural networks are *spatially* iterative neural networks, which have several layers of *hidden neuron units* between the input and output neuron layers (see Fig. 1). the weight updating for the hidden layers adopts the mechanism of back-propagated corrective signal from the output layer.

Retrieving Phase: The system dynamics in the retrieving phase of an L-layer perceptron can be described by the following spatially iterative equations, with the iteration index l denoting the *spatial* (layer) iteration:

$$u_i(l+1) = \sum_{j=1}^{N_l} w_{ij}(l+1)a_j(l)$$

$$a_i(l+1) = f(u_i(l+1) + \theta_i(l+1)) = f_i(l+1) \tag{17}$$

where $1 \le i \le N_{l+1}$, $0 \le l \le L-1$, and the nonlinear function f is nondecreasing and differentiable. For simplicity, the external inputs $\{\theta_i(l+1)\}$ are often treated as special modifiable synaptic weights $\{w_{i,0}(l+1)\}$ which have clamped inputs $a_0(l) = 1$.

Learning Phase: The learning phase of an L-layer multilayer perceptron follows an iterative gradient descent approach [54], [55], [64], [76], [77]. Given the pth pair of input/target training patterns, $\{\alpha_i^{(p)}, i = 1, \cdots, N_0\}$, $\{t_j^{(p)}, j = 1, \cdots, N_L\}$, our goal; is to iteratively choose a set of $\{w_{ij}(l), \forall l\}$ for all layers so that the mean squared error between the actual output activations and target activations can be minimized. To be more specific

$$w_{ij}(l) \Leftarrow w_{ij}(l) - \eta\delta_i(l)f_i'(l)a_j(l-1) \tag{18}$$

where $1 \le l \le L$, and $f_i'(l)$ is the derivative of $f_i(l)$ with respect to $u_i(l)$. The back-propagated corrective signal $\delta_i(l)$ can be recursively calculated as shown below. For $L-1 \ge l \ge 1$

$$\delta_i(l) = \sum_{j=1}^{N_{l+1}} \delta_j(l+1)f_j'(l+1)w_{ji}(l+1). \tag{19}$$

Note that the backward initialization error signal on the top layer $\delta_i(L)$ is defined as $\delta_i(L) = -(t_i - \beta_i)$.

III. Applications of ANN's to the Robotic Processing

This section discusses *how to use the above neural networks for robotic processing*. There are three levels of processing in a hierarchical robotic processing system, task planning, path planning, and path control [68]. In the task

planning level the robot will receive the instructions about task plans and manage the information (e.g., target, depth, obstacle locations) about the workspace. The path planning is to produce a sequence of desired path trajectory (e.g., robot positions, arm orientations). The path control is to generate the necessary motor commands (torques and forces) in the joint coordinates to drive the robot or arms to follow the desired trajectory. The computational requirement of the robotic processing is very demanding. At the level of task planning, some primitive image/vision analysis are required to obtain the three-dimensional (3-D) information. At the level of path planning, an optimal path in a workspace has to be selected by using optimization techniques. For the path control level, real-time computing of inverse dynamics and kinematics must be provided.

In this section, neural network techniques for processing the various algorithms in the three levels of robotic processing are discussed. The approach is to formulate each algorithm either as an optimization problem or a pattern association problem. The examples include stereo vision in task planning [49], [70], [71], [80], autonomous robot path planning [31], [74], manipulator position control [58], [75], robotic voluntary movement, and sensor/motor control [32], [42], [62]. Our aim is to demonstrate how these algorithms and some other typical operations in robotic processing can be solved by the neural networks.

A. Task Planing Using Neural Networks

A critical step in the task planning is the determination of the depth information. Traditionally, the sonar sensors have been very popular for this purpose [30]. However, the stereo matching technique is a more reliable but computationally costly alternative. It recovers 3-D depth information from two images taken from two cameras. Usually, the geometry of the imaging system (e.g., the camera postions and their orientations) are known. The main task in the stereo vision then hinges upon the matching between the feature primitives of the two images. This is called the *correspondence problem*, which consists of the detection of the feature primitives (e.g., edges [70]) and the determination of the valid matches of the feature primitives. As discussed next, the latter task may be formulated as a constrained optimization problem [49], [70], [71], [80].

Neural Networks for Constrained Optimization: Consider the following constrained optimization problem [45]:

$$\text{minimize} \quad \phi(\boldsymbol{a})$$
$$\text{subject to the constraints} \quad P_i(\boldsymbol{a}) = 0, \qquad i = 1, 2, \cdots, q \tag{20}$$

where ϕ is a continuous function of n-dimensional vector $\boldsymbol{a}$ (continuous or discrete) and $\{P_i(\boldsymbol{a})\}$ are nonnegative and continuous functions. The above constrained optimization problem may be approximated by an unconstrained optimization problem, which involves minimizing a new energy (or penalty) function [45]

$$\text{minimize} \quad E' = \phi(\boldsymbol{a}) + \sum_{i=1}^{q} c_i P_i(\boldsymbol{a}) \tag{21}$$

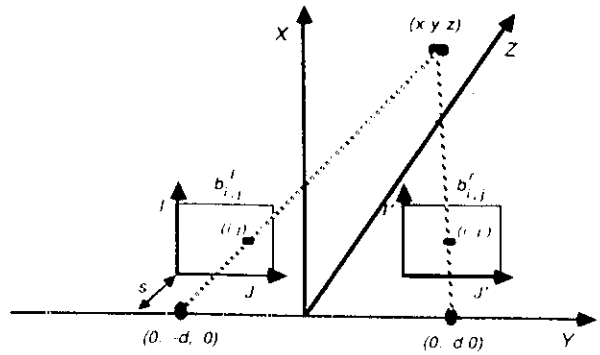


Fig. 3. A standard stereo matching problem involves two camera with focal length s and their focal points lying at $(0, \pm d, 0)$.

where $\{c_i\}$ are large positive constants. This naturally leads to the optimization neural network processing and the synaptic weights $\{w_{ij}\}$ and the external inputs $\{\theta_i\}$ can be derived from the (Liapunov) energy function.

Neural Network Formulation for Stereo Matching: As shown in Fig. 3, the stereo matching problem involves two cameras with focal length s and their focal points lying at $(0, \pm d, 0)$, respectively. The lines of sight of the cameras are assumed to be parallel to the z-axis. A point (x, y, z) appears at (i, j) pixel of the left image coordinate and at (i', j') pixel of the left image coordinate. Without loss of generality, let us assume that the epipolar scanline is along the horizontal (J-axis) direction, we can have $i = i'$, and the search for candidate match for a point can be limited to the same row in the other image. The value $(j - j')$ is caled the *disparity* value between two matching points. The depth information z can be generated from the disparity value, i.e.,

$$z = \frac{2sd}{(j - j')}.$$

Suppose that there are two $M \times N$ primitives extracted image intensity arrays, $\{b^l_{i,j}\}$ and $\{b^r_{i,j}\}$ as taken by the (left and right) cameras. A 3-D binary matching data array $\{a_{i,j,k}, 1 \le i \le M, 1 \le j \le N, 0 \le k \le D\}$ may be defined to represent the status of matching [49], [80]. When the $a_{i,j,k}$ is 1, this means that the disparity value is k at the point (i, j). In this discussion, the maximal disparity value is limited to D. The goal of the correspondence problem is to minimize the following cost function ϕ:

$$\phi = \phi_1 + \phi_2$$
$$= \sum_{i=1}^{M} \sum_{j=1}^{N} \sum_{k=0}^{D} (b^l_{i,j} - b^r_{i,j+k})^2 a_{i,j,k} + \lambda \sum_{i=1}^{M} \sum_{j=1}^{N} \sum_{k=0}^{D} \sum_{(i',j') \in \Psi} (a_{i,j,k} - a_{i',j',k})^2 \tag{22}$$

with the constraint

$$P = \sum_{i=1}^{M} \sum_{j=1}^{N} \left(\sum_{k=0}^{D} a_{i,j,k} - 1 \right)^2 = 0 \tag{23}$$

where

1) ϕ_1 is a measurement of how two images are matched after alignment in a least squares sense. The symbol $\oplus$ denotes that

$$t_{a\oplus b}=\begin{cases} t_{a+b}, & \text{if } 0\le a+b\le N \\ 0, & \text{otherwise.} \end{cases}$$

2) ϕ_2 is a measurement of the continuity of the depth value [49], [80], where λ is a proper weighting constant and Ψ denotes the the neighborhood around the pixel (i, j), defined by a square window centered at (i, j).

3) The constraint in (23) is introduced to assure the *uniqueness* property is preserved. That is any point from each image can assume one and only one depth value [9], [49].

Following (21)–(23), the constrained problem can be reformulated as an unconstrained problem i.e.,

$$\min_{a_{i,j,k}} E' = \phi + cP. \tag{24}$$

The Hopfield optimization network or Boltzman machine can be used to solve the above unconstrained problem. Each matching data element $A_{i,j,k}$ can be regarded to be an activation value of one neuron. As discussed in Section I-B, the synaptic weights $w_{i,j,k,l,m,n}$ and the external input $\theta_{i,j,k}$ of the neural network can be derived by equating the energy (penalty) function in (24) to the Liapunov energy function E in (4) [27], [80], i.e.,

$$E=E'=-\frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{k=0}^{D}\sum_{l=1}^{M}\sum_{m=1}^{N}\sum_{n=0}^{D} w_{i,j,k,l,m,n}\cdot a_{i,j,k}a_{l,m,n}-\sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{k=0}^{D}\theta_{i,j,k}a_{i,j,k}.$$

B. Path Planning Using Neural Networks

The problem of path planning for a robot can be stated as follows: Given the locations of a set of obstacles and the initial position and the final target of the robot, to find a continuous path from the initial position to the target while circumventing the obstacles along the way. Two types of planning are often encountered: one is called *global planning*, which derives the optimal path from the overall topographic map. The other is called *local planning*, which relies on information available only to the line of sight of the robot sensors or cameras from particular perspectives. We note that human beings perform very well in the path planning through a two-stage anticipatory planning [31]: First they recall the complete global environment from the local line of sight information via associative retrieval. Then they recursively search for the optimal path from the retrieved global information. In a manner very much similar to that of the human being, artifical neural networks may be adopted to implement the operations in both stages.

1) Global Terrain Retrieval via Pattern Association: The robot navigation area can be divided into 2-D or 3-D grid cells. A binary value may be assigned to each grid cell to indicate the clearness status. Namely, a value "1" implies that the cell is fully occupied, while "−1" implies that the cell is unoccupied. Continuous values between −1 and 1 may also be used to indicate partial occupation of a grid cell or any ambiguity due to vision/sensor processing. The values of the grid cells can be trained and recalled using associative retrieval neural networks. When a robot "perceives" an environment similar to one previously trained, the neural network accepts the (continuous) values of the local grid cells to retrieve a best fitting global terrain features based on the trained data. Possible candidates for this associative retrieval task are the Rumelhart's *memory/learning modules* [50] and multilayer perceptrons [64]. The reasons for adopting such neural networks are that they works well with continuous input values and they exhibit good fault tolerance capabilities in the retrieval.

2) Boltzmann Machine for Optimal Path Finding: Once the information about the global grid cells is retrieved then the neural network is ready to plan an optimal path to reach a remote goal. In order to find the global optimal path, a constrained optimization formulation can be adopted and neural networks applied. The key is to define a cost function ϕ. It is a function of several important variables, e.g., the distance between the current location and the starting position of the robot or the distance between the current location to the goal, etc. Some constraints need to be satisfied, e.g., the path should not be allowed to pass through the grid cells with high clearness values. The robot will stop and start a new path planning process when an unanticipated obstacle is sensed before it reaches the final target position. In order not to be trapped in local optima, simulated annealing techniques such as Boltzmann machine can be adopted. It is important to provide a good (although not optimal) path as an initial state for the optimization process. The recalled global map is very useful to generate such a preliminary path. Based on the map, those unlikely cells, e.g., cells outside the boundary or cells occupied with obstacles, can be ruled out. Then the cells with a minimum traversal distance of the path may be selected to be the preliminary path.

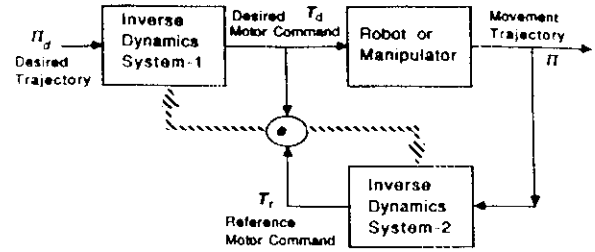


Fig. 4. A simplified schematical diagram for the feed-forward controller

C. Path Control Using Neural Networks

After the determination of the desired trajectory in the path planning level, the task of path control is to generate the motor control signal (torques and forces) so that the robot may be driven to follow the trajectory.

1) Inverse Dynamics via Multilayer Perceptron: Based on some physiological model [32], [59], the dynamic movement of a robot or a manipulator can be controlled by a feed-forward controller, as shown in Fig. 4. It consists of three major components: two *identical* inverse dynamics systems ($IDS_1 = IDS_2$) and the robot (or manipulator).

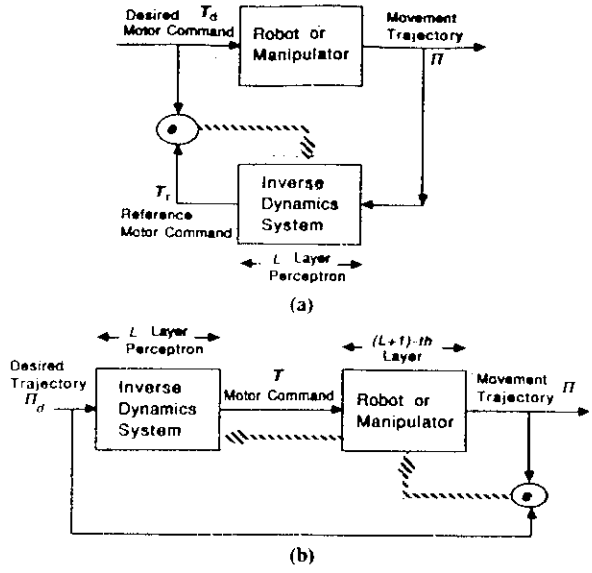


Fig. 5. (a) Generalized learning configuration. (b) Specialized learning configuration. Note that the *IDS* can be modeled by an L layer perceptron.

The IDS_1 receives the desired trajectory information Π_d and produces the corresponding desired motor command T_d to drive the robot so that the actual movement of the robot (denoted as Π) may follow as closely as possible the desired trajectory Π_d. The IDS_2, on the other hand, takes the actual movement Π and produces the reference motor command T_r as shown in Fig. 4.

To efficiently train the *IDS* to implement the feed-forward controller as shown in Fig. 4, a two-stages learning procedure can be adopted [32], [58]. The first stage is called *generalized learning* with its configuration shown in Fig. 5(a), the second stage is called *specialized learning* with a different configuration shown in Fig. 5(b). In the generalized learning, a set of desired motor commands, denoted as $\{T_d\}$, are used to drive the robot and the set of resulting trajectories is denoted as $\{\Pi\}$. Then the *IDS* receive $\{\Pi\}$ as input and yield a set of reference motor commands, denoted as $\{T_r\}$. The goal of the generalized learning is to minimize the errors between $\{T_r\}$ and $\{T_d\}$ in the least square sense [59]. After the *IDS* is well trained, if a real input Π' is sufficiently close to one trajectory in the set $\{\Pi\}$, the controller should be able to retrieve a proper motor command $\hat{T}$, making the actual movement $\hat{\Pi}$ closely follows Π.

Due to the lack of knowledge about the operating range of the desired motor commands $\{T_d\}$, an unnecessarily large set of $\{T_d\}$ for the training may have to be used. This difficulty can be overcome by incorporating a specialized learning stage into the controller system (see Fig. 5(b)), in which the *IDS* is trained based on the desired trajectory $\{\Pi_d = [\Pi_1^{(d)}, \cdots, \Pi_m^{(d)}]\}$ and outputs the appropriate motor commands $\{T\}$ to drive the robot. The actual movement trajectory of robot is a function of T, denoted as $\Pi(T) = [\Pi_1, \cdots, \Pi_m]$. When the operating points of the system change or when new training patterns are added, it should be adequate to use specialized learning to fine-tune the system.

A severe weakness in the specialized learning is that in the initial step, the training of *IDS* may be very inefficient due to the lack of knowledge about the dynamic model of the robot. Therefore, it is advantageous to properly combine the generalized learning and the specialized learning.[2] For example, it is possible to first perform the generalized learning until the dynamic model of the robot is approximately learned, then the specialized learning follows to fine-tune the *IDS*.

Multilayer Perceptrons for Generalized Learning: An L-layer neural network can be used to implement the *IDS* in a generalized learning system [59]: An input $T_d = [T_1^{(d)}, \cdots, T_n^{(d)}]$ is selected an applied to the robot to obtain a corresponding Π, and the network is trained to reproduce $T_r = [T_1^{(r)}, T_2^{(r)}, \cdots, T_n^{(r)}]$ at its output from Π (see Fig. 5(a)). The mathematical formulation for the updating of weights $w_{ij}(l)$ using the back propagation learning as given in (18) and (19). Note that the weights training here is based on the least squares errors of $|T_d - T_r|^2$, although the real objective of the robot training should have been minimizing $|\Pi_d - \Pi|^2$.

Multilayer Perceptrons for Specialized Learning: The same L-layer neural network can be used to implement the *IDS* in a specialized learning [59]: Referring to Fig. 5(b), the dynamic model of the robot can be regarded as an additional layer, i.e., the $(L + 1)$th layer. However, the back propagation learning may not be applicable to this last layer, since the layer has no synaptic connections defined in the conventional sense. Instead, a modified back-propagation formula was proposed [58]:

$$\delta_i(L) = \sum_j \delta_j(L+1) \frac{\partial \Pi_j(T)}{\partial T_i}$$

$$\delta_j(L+1) = -(\Pi_j^{(d)} - \Pi_j)$$

where $\Pi_j(T)$ denotes the jth element of the robot movement trajectory. In case the dynamic model of the robot is unknown, the partial derivative can be approximated as

$$\frac{\partial \Pi_j(T)}{\partial T_i} \approx \frac{\Pi_j(T + \Delta T_i I_i) - \Pi_j(T)}{\Delta T_i}$$

where $I_i = [0, 0, \cdots, 1, 0, \cdots, 0]$. For the training of the remaining L layers of the multilayer perceptron, the conventional back propagation learning algorithm (see (18) and (19)) can be adopted.

2) Sensor/Motor Maps Using Competitive Learning: It is important to apply robots to an unforseen environment. For example, various obstacles may enter or leave the working area of the robot and need to be avoided. This can be solved by providing a visual map (sensor/motor topographic map) of the objects in the environment to the robot controller.

Neural architecture with self-organized competitive learning (see (15)) can be used to control adaptive sensory-motor coordination [42]. In the learning phase, visual input signals about the object are processed and combined into a target map through modifiable weights, and producing computed motor signals. The errors between the actual motor signals and the motor signals computed from the visual input are used to

[2] By switching back and forth between the two learning stages, it can also avoid the system to be trapped by local minima [58].

incrementally change the weights to make the later computed motor signals closer to the actual computed signals. After the network is trained, in the retrieving phase the learned sensory–motor correlation is used to recognize and manipulate objects which are similar to those that were experienced in the first stage. This neural architecture is composed of motor map representations interleaved within a sensory topography. This allows any number of topographic sensory inputs to be mapped onto any number of motor outputs.

Darwin networks is a simulated automaton made up of many subnetworks [62]. Inputs to the automaton are from a simulated eye which scans a 2-D input array under control of opponent pair muscles. This eye has a large but low-resolution outer visual field and a smaller, high-resolution inner visual field on its simulated retina. Inputs are also provided by a multiple-jointed arm that can reach and feel objects presented on the input field. Darwin networks is first trained to track objects presented on the input array by coordinated movement of eye muscles. The error signal to learn this task is the distance between the position of the object on the retina and the center or fovea of the retina. Darwin networks is then trained to reach out, touch, and feel around the border of objects using self-organized competitive learning [46].

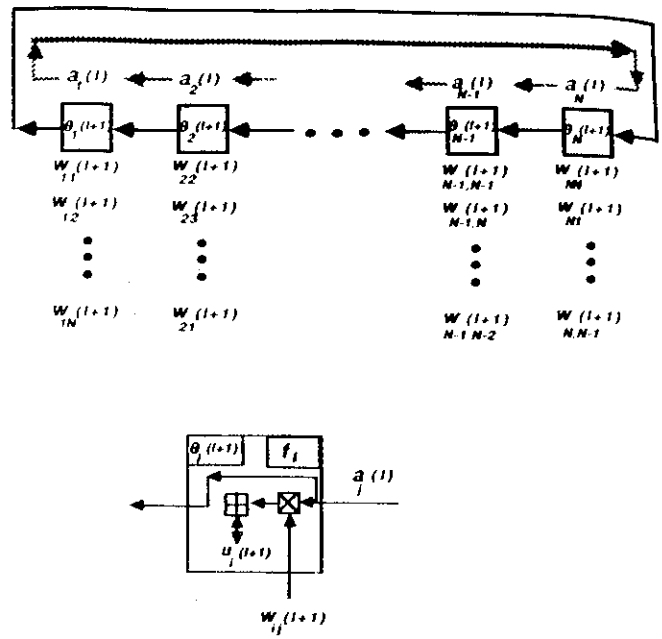


Fig. 6. The ring systolic architecture for consecutive MVM in the retrieving phase at $l+1$th iteration.

IV. Ring Systolic Architecture for Neural Networks

It is shown that operations in both the retrieving and learning phases of most iterative ANN models can be formulated as *consecutive matrix–vector multiplication*, *consecutive vector–matrix multiplication*, or *outer-product updating* problems. In terms of the array structure, all these formulations lead to a same universal ring systolic array architecture. In terms of the functional operations, all these formulations calls for a MAC (multiply and accumulation) processor [41].

A. Ring Systolic Design for the Retrieving Phase

Consecutive MVM in the Retrieving Phase: The system dynamics in the retrieving phase of the generic iterative ANN model can be formulated as a consecutive *matrix–vector multiplication* (MVM) problem interleaved with the nonlinear activation function (see (1) and (2)). Without loss of generality and to facilitate the homogeneous architectural design, it is first (which will be relaxed later) assumed that all the iterations in the iterative ANN model have uniform size of N neural units, which is true for single-layer feedback networks, where $\{w_{ij}(l) = w_{ij}\}$, and $\{\theta_i(l) = \theta_i\}$.

$$\begin{aligned} \boldsymbol{u}(l+1) &= \boldsymbol{W}(l+1)\boldsymbol{a}(l) \\ \boldsymbol{a}(l+1) &= f[\boldsymbol{u}(l+1), \theta(l+1)] \end{aligned} \tag{25}$$

where $\boldsymbol{u}(l) = [u_1(l), u_2(l), \cdots, u_N(l)]^T$, $\boldsymbol{a}(l) = [a_1(l), a_2(l), \cdots, a_N(l)]^T$, $\theta(l) = [\theta_1(l), \theta_2(l), \cdots, \theta_N(l)]^T$ and $\boldsymbol{W}(l) = \{w_{ij}(l)\}$. The $f[\boldsymbol{x}]$ operator performs the nonlinear activation function f on each element of the vector $\boldsymbol{x}$.

This uniform size consecutive MVM formulation leads to a ring systolic architecture as shown in Fig. 6 [40], [41]. The pipelining period of this design is 1, which implies 100-percent utilization efficiency [38]. A minor disadvantage of the design is that the (global) spiral communication link is required.

Systolic Processing of Consecutive MVM: At the $(l+1)$th iteration of the retrieving phase, each processor element (PE), say the ith PE, can be treated as a neuron, and the corresponding incoming synaptic weights $(w_{i1}(l+1), w_{i2}(l+1), \cdots, w_{iN}(l+1))$ are stored in the memory of the ith PE in a circularly shift-up order (by $i-1$ positions). The operations at the $(l+1)$th iteration can be described as follows (see Fig. 6):

1) Each neuron activation value $a_i(l)$, created at ith PE, is multiplied with $w_{ij}(l+1)$. The product is added to the accumulator $u_i(l+1)$, which has initial value set to be equal to zero (or $\theta_i(l+1)$ in some networks). After the MAC operation, $a_i(l)$ will move counterclockwise across the ring array and visit each of the other PE's once in N clock units.

2) When $a_i(l)$ arrives at the ith PE, it is multiplied with $w_{ij}(l+1)$ and the product is added to $u_i(l+1)$ according to (1).

3) After N clock units, the accumulator $u_i(l+1)$ collects all the necessary products.

4) One more clock is needed for $a_i(l)$ to returns to the ith PE and the processor is ready for the nonlinear activation operation f_i (see (2)) to create $a_i(l+1)$ for the next iteration. (Note that N clocks are required to perform the winner-take-all nonlinear mechanism by cycling all the $u_i(l+1)$ to determine the winner.)

The above procedure can be executed in a fully pipelined fashion. Moreover, it can be recursively executed (with increased l) until L iterations are completed.

B. Ring Systolic Design for the Learning Phase

There are two types of operations required for most weight updating methods: 1) the *outer product updating* (OPU), and 2) the consecutive *vector–matrix multiplication* (VMM).

Both types can be efficiently implemented by the ring systolic ANN based on the same memory storage, processing hardware, and array configuration in the retrieving phase.

1) Operations in the Learning Phase: Given the weight value $w_{ij}(l+1)$ of the previous learning recursion, 1) the new weight values can be calculated based on the OPU; 2) and the back-propagated corrective signal $\delta_i(l)$ of ith iteration can also be computed based on the consecutive VMM operation.

OPU in the Learning Phase: In general, an additive updating formulation of the weights at $(l+1)$th iteration can be summarized by the following OPU equation [65]:

$$w_{ij}(l+1) \Leftarrow w_{ij}(l+1)+\Delta w_{ij}(l+1)$$

$$= w_{ij}(l+1)+g_i(l+1)\cdot h_j(l+1) \text{ or}$$

$$\boldsymbol{W}(l+1) \Leftarrow \boldsymbol{W}(l+1)+\boldsymbol{g}(l+1)\boldsymbol{h}^T(l+1)$$

where $\boldsymbol{g}(l+1) = [g_1(l+1), g_2(l+1), \cdots, g_N(l+1)]^T$ and $\mathbf{h}(l+1) = [h_1(l+1), h_2(l+1), \cdots, h_N(l+1)]^T$. To be more specific

$g_i(l+1)$	$h_j(l+1)$	Learning
β_i	$\eta\beta_j$	Hebbian
$t_i - \beta_i$	$\eta\beta_i$	Delta
β_i	$\eta(\alpha_j - w_{ij})$	Competitive
$\delta_i(l+1)f_i'(l+1)$	$-\eta a_j(l)$	Generalized Delta

Consecutive VMM in the Learning Phase: The back-propagation rule can be formulated as a consecutive VMM operations [41]

$$\delta_i(l)=\sum_{j=1}^{N} g_j(l+1)w_{ji}(l+1)$$

or

$$\boldsymbol{d}^T(l)=\boldsymbol{g}^T(l+1)\boldsymbol{W}(l+1) \qquad (26)$$

where $\boldsymbol{d}(l) = [\delta_1(l), \delta_2(l), \cdots, \delta_N(l)]^T$.

2) Systolic Processing in the Learning Phase: The ring systolic array derived for the retrieving phase can be easily adapted to execute in parallel both the OPU and consecutive VMM in the learning phase [41].

Systolic Processing for the OPU: The operations of the OPU in the ring systolic ANN can be briefly described as follows (see Fig. 7):

1) The value of $g_i(l+1)$ is computed and stored in the ith PE. The value $h_j(l+1)$ produced at the jth PE will be cyclically piped (leftward) to all other PE's in the ring systolic ANN during the N clocks.

2) When $h_j(l+1)$ arrives at the ith PE, it is multiplied with the stored value $g_i(l+1)$ to yield $\Delta w_{ij}(l+1)$, which will be added to the old weight $w_{ij}(l+1)$ of the previous recursion to yield the updated $w_{ij}(l+1)$. Note that the old weight data $\{w_{ij}(l+1)\}$ are retrieved in a circularly-shift-up sequence, just like what used in the retrieving phase.

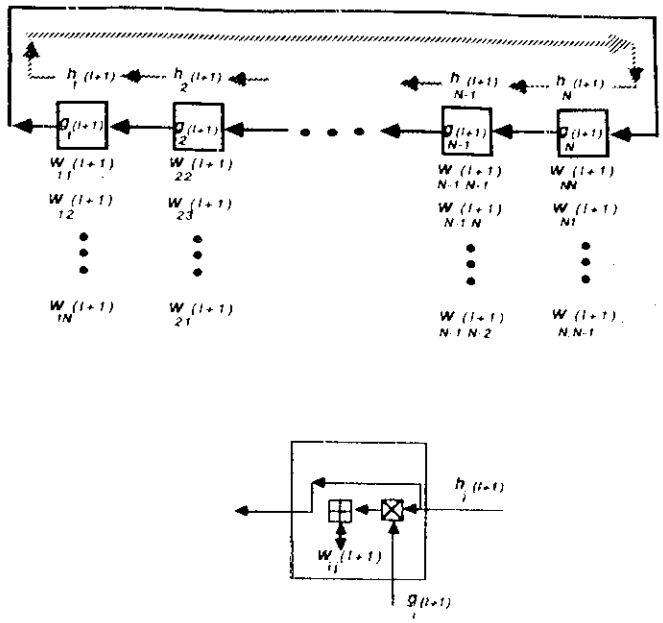


Fig. 7. The ring systolic architecture for OPU in the learning phase.

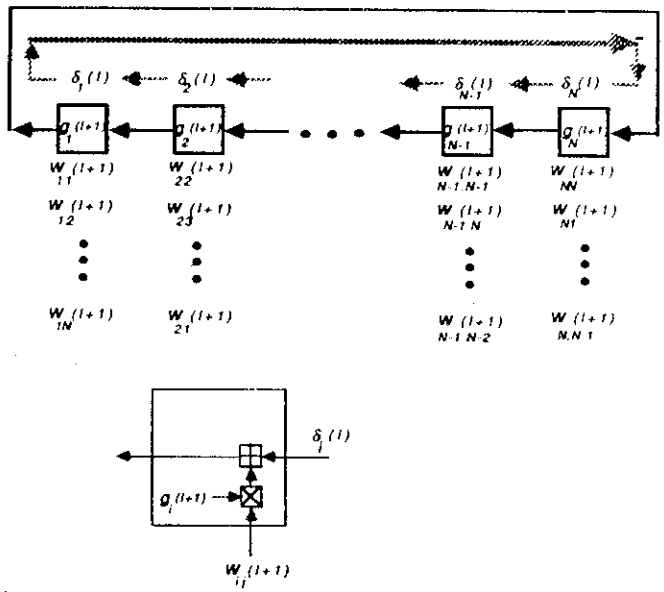


Fig. 8. The ring systolic architecture for consecutive VMM in the learning phase.

3) After N clocks, all the $N \times N$ new weights at $(l+1)$th iteration $\{w_{ij}(l+1)\}$ are generated. The ring systolic ANN is now ready for the weight updating of the next iteration (l).

Systolic Processing for the Consecutive VMM: The operations of the consecutive VMM in the ring systolic ANN can be briefly described as follows (see Fig. 8):

1) The signal $g_j(l+1)$ and the value $w_{ji}(l+1)$ are available in the jth PE at $(l+1)$th iteration. The value $w_{ji}(l+1)$ is then multiplied with $g_j(l+1)$ at jth PE.

2) The product is added to the newly arrived accumulator $\delta_i(l)$. (The parameter $\delta_i(l)$ is initiated at the ith PE with zero initial value and circularly shifted leftward across the ring array.)

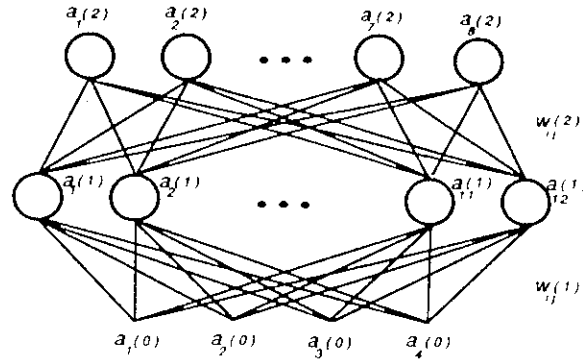


Fig. 9. A two-layer perceptron with configuration (4–12–8)

3) After N such accumulation operations, the accumulator $\delta_i(l)$ will return to ith PE after accumulating all the products

$$\delta_i(l) = \sum_{j=1}^{N} g_j(l+1) w_{ji}(l+1).$$

C. Partitioning Scheme for Large or Nonuniform Networks

Generally speaking, in an overall array architecture system, one seeks to maximize the following performance indicators: effective array configuration, flexibility on problem partitioning, fault-tolerance to improve system reliability, and programmability for adequate software support [38].

In case of large size of neural networks with smaller number of available PE's, it is important to provide hardware and/or software support for an efficient partitioning scheme, which allows large problems to be decomposed into smaller subproblems to be solved on the array. The tasks of several neural networks can be assigned to share the same PE without changing the computation/communication strategy. Based on this approach, this architecture can be easily adapted to nonuniform multilayer perceptron, where each layer has very different number of neurons. By appropriately assigning the tasks of several (equal number of) neurons at the same layer to one specific PE, the computational load of each layer can be uniformly distributed to the N PE's disregarding the number of neurons in each layer. This also guarantees the fully pipelining efficiency.

Without loss of generality, an example is given to illustrate the partitioning scheme for nonuniform networks. We will show that a ring architecture with 4 PE's can be used to implement the retrieving/learning phase of a two-layer perceptron shown in Fig. 9 with configuration (4–12–8, 4 input neurons, 12 hidden neurons, and 8 output neurons). If the number of neurons are not exactly equal to the multiple of 4, it is always possible to artifically pad a suitable number of pseudo (no-operation) neural units (i.e., the weights connected to the units are set to be zero) to match the size.

1) Design for the Retrieving Phase: The consecutive MVM operations can be partitioned based on the *locally parallel globally sequential* (LPGS) scheme [38]:

1) the tasks of the 12 neurons in the first (hidden) layer is uniformly distributed to the 4 PE's, and the associated weights are appropriately arranged as before. Instead of cycling once in the ring array, the four inputs $\{a_i(0)\}$ are cycling three times to generate all the 12 activation values $\{a_i(1)\}$, i.e., $\{a_i(1), i = 1, \cdots, 4\}$, then $\{a_i(1), i = 5, \cdots, 8\}$, and finally $\{a_i(1), i = 9, \cdots, 12\}$ (see Fig. 10(a)).

2) The resulting 12 $\{a_i(1)\}$ will cycle in the ring (4-by-4 sequentially) six times to generate the eight activation values $\{a_i(2)\}$ of the output layer. The first three times to generate the first four activation values $\{a_i(2), i = 1, \cdots, 4\}$, and the second three times to generate the rest of activation values $\{a_i(2), i = 5, \cdots, 8\}$ (see Fig. 10(b)).

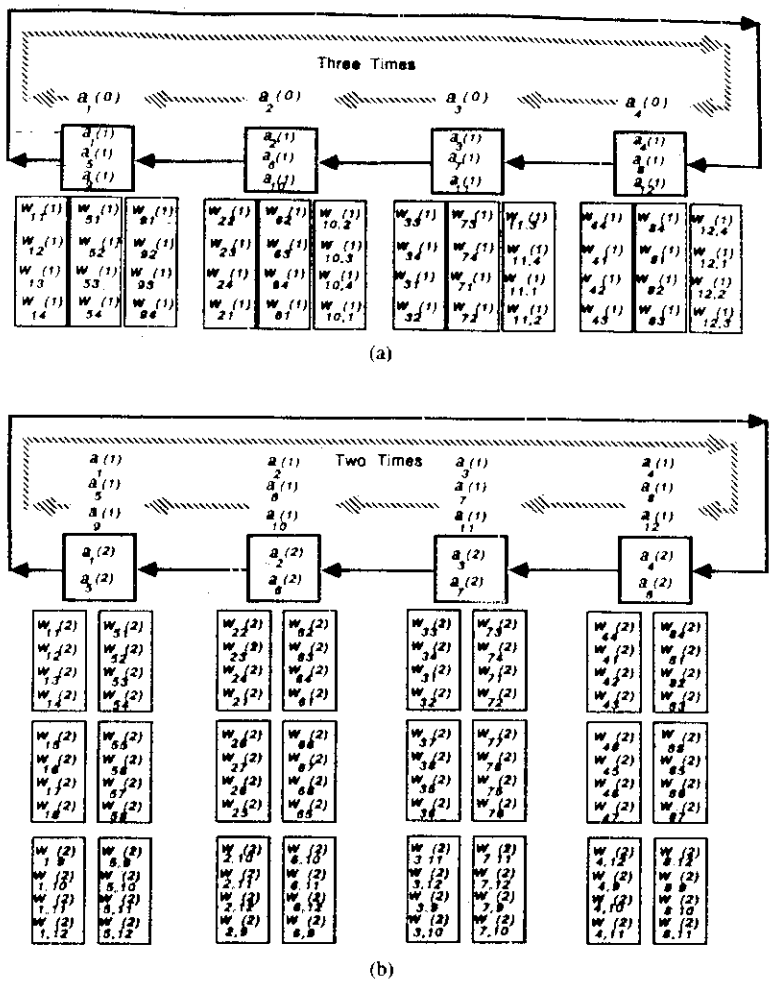
2) Design for the Learning Phase:

Systolic Processing of the OPU: The operations of the OPU for the nonuniform multilayer perceptron can again be implemented in the ring systolic ANN. Take for example, the same 4–12–8 two layer perceptron as shown in Fig. 9. Assume that the eight $\{g_i(2) = \delta_i(2) f_i'(2)\}$ and the twelve $\{h_j(2) = \eta a_j(1)\}$ are available at the corresponding PE, so are the twelve $\{g_i(1) = \delta_i(1) f_i'(1)\}$ and the four $\{h_j(1) = \eta a_j(0)\}$. The operations can be briefly described as follows (see Fig. 11):

1) The twelve $\{h_j(2)\}$ will cycle in the ring array in three sequential batches, $\{h_j(2), j = 1, \cdots, 4\}$, $\{h_j(2), j = 5, \cdots, 8\}$, and $\{h_j(2), j = 9, \cdots, 12\}$, to first update all the weights associated with the first four neurons in the output (second) layer $\{w_{ij}(2), i = 1, \cdots, 4, j = 1, \cdots, 12\}$. Then these three sequential batches are again cycled in the ring to update the weights associated with the last four neurons in the output layer $\{w_{ij}(2), i = 5, \cdots, 8, j = 1, \cdots, 12\}$ (see Fig. 11(a)).

2) The same four $\{h_j(1)\}$ will cycle in the ring array three times to update all the weights associated with the twelve neurons in the hidden (first) layer. Each cycling of $\{h_j(1)\}$ will update the weights associated to four neurons in the hidden layer, i.e., $\{w_{ij}(1), i = 1, \cdots, 4, j = 1, \cdots, 4\}$, $\{w_{ij}(1), i = 5, \cdots, 8, j = 1, \cdots, 4\}$, and $\{w_{ij}(1), i = 9, \cdots, 12, j = 1, \cdots, 4\}$ (see Fig. 11(b)).

Systolc Processing of the Consecutive VMM: At the same time when the operations of the OPU are performed, the consecutive VMM can be executed in parallel to compute the $\{\delta_i(1)\}$. The operations can be briefly described as follows



(a)

(b)

Fig. 10. The ring architecture with 4 PE's can be used to implement the retrieving phase of this two-layer perceptron. (a) The forward retrieving operations in the first (hidden) layer. (b) The forward retrieving operations in the second (output) layer.

(see Fig. 12):

1) Along with the cycling of each $h_j(2)$ in the ring array for the OPU operation, there are also an accumulator $\delta_j(1)$ cycling in the ring array in three sequential batches, $\{\delta_j(1), j = 1, \cdots, 4\}$, $\{\delta_j(1), j = 5, \cdots, 8\}$, and $\{\delta_j(1), j = 9, \cdots, 12\}$, to accumulate the first half the back-propagated corrective signals, i.e., at the same time as the updating of all the weights associated with the first four neurons in the output layer is done (see Fig. 12(a)), and

$$\delta_i(1) = \sum_{j=1}^{4} g_j(2) w_{ji}(2).$$

2) When the second three-time cycling are performed to update the weights associated with the last four neurons in the output layer, the $\{\delta_j(1)\}$ are again accumulated to form (see Fig. 12(b))

$$\delta_i(1) = \sum_{j=1}^{8} g_j(2) w_{ji}(2).$$

D. Implementation Considerations of Neural Processing Units

Time-Efficient Design Based on a Parallel Array Multiplier: As discussed in Sections IV-A and IV-B, most of the computations involved in the neural processing units are MAC or multiplication (e.g., the calculations of $g_i(l + 1)$, and $h_j(l + 1)$, and $r_{ji}(l + 1)$) operations, so for a *time-efficient* dedicated design, a parallel array multiplier (e.g., Baugh–Wooley multiplier [2], [10]) should be favorably considered.

There is concern about using digital hardware for the nonlinear sigmoid function which is required in many continuous-valued ANN models. Justifed by the simulations, the nonlinear sigmoid function can be well approximated by a piecewise-linear function with 8–16 segments [47]. This again calls for the MAC processor, which greatly simplifies the hardware complexity.

Area-Efficient Design Based on Cordic Processor: For an area-efficient dedicated digital VLSI implementation of the neural processing units, a Cordic processer, which uses about 1/3 silicon area of an array multiplier might be a good

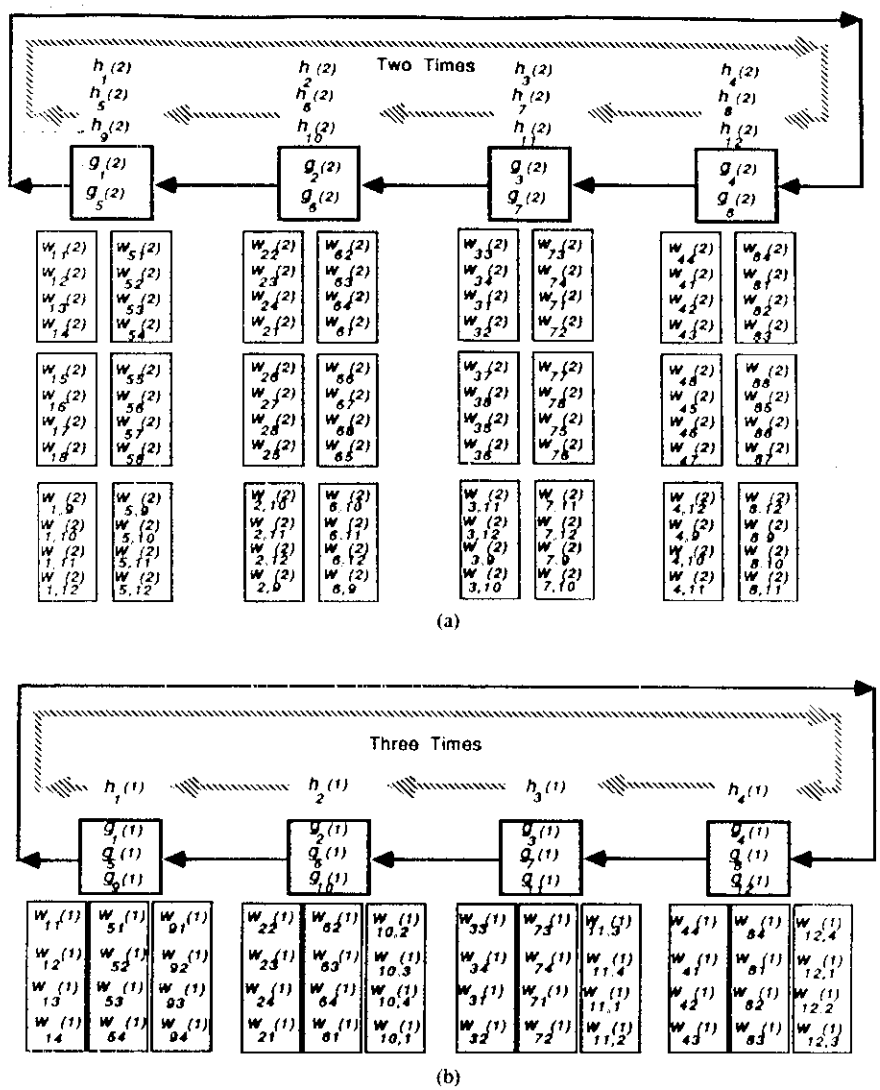


Fig. 11 The ring architecture for the implementation of OPU in a two-layer perceptron with configuration (4-12-8). (a) The OPU operations is the second (output) layer. (b) The OPU operations in the first (hidden) layer.

alternative [2]. A Cordic scheme is an iterative method based on bit-level shift-and-add operations. It is especially suitable for computing a class of rotations [38]. A 2-D vector $v = [x, y]$ can be rotated be an angle α by using a rotation operator $R_\alpha v' = R_\alpha v$.

In a linear mode, the Cordic can be used for MAC (multiple and accumulation) operations, although somewhat slower than the array multiplier. Given x, y, and α, we can get the Cordic output $y + x\alpha$ in the linear mode which is needed for the MAC operations. The key advantage of Cordic is that it may implement the sigmoid activation function by setting the Cordic in the hyperbolic mode, where the inputs are set as $v = [1, 1]$ and $\alpha = u/2$. For a more detailed design, the readers should refer to [2], [28].

V. Conclusion

Due to its robustness and adaptiveness, neural network processing can be very useful to all levels of robotic applications. For real-time processing performance, the neural network architectures for robotic processing will require massive parallel processing. Fortunately, today's VSLI and CAD technologies facilitate practical and cost-effective implementation of large-scale computing networks. In order to fully exploit VLSI's strength, a programmable systolic neural network architecture is proposed in this paper. Thanks to the versatility of the systolic design, most neural network models for robotic applications can be efficiently implemented. One critical issue which requires a closer investigation is the convergence property for both the retrieving and learning phases of the neural networks. For example, the number of hidden units in a multilayer perceptron must be sufficient to provide the discriminating capability required by the given application, but an excessively large number of synaptic weights may lead to costly and unreliable training. Therefore, it s very desirable to have an *a prior* estimate of an optimal number of hidden neurons. For this and to better understand the convergence property, it is essential to develop a theoretical footing based on system theory and numerical analysis

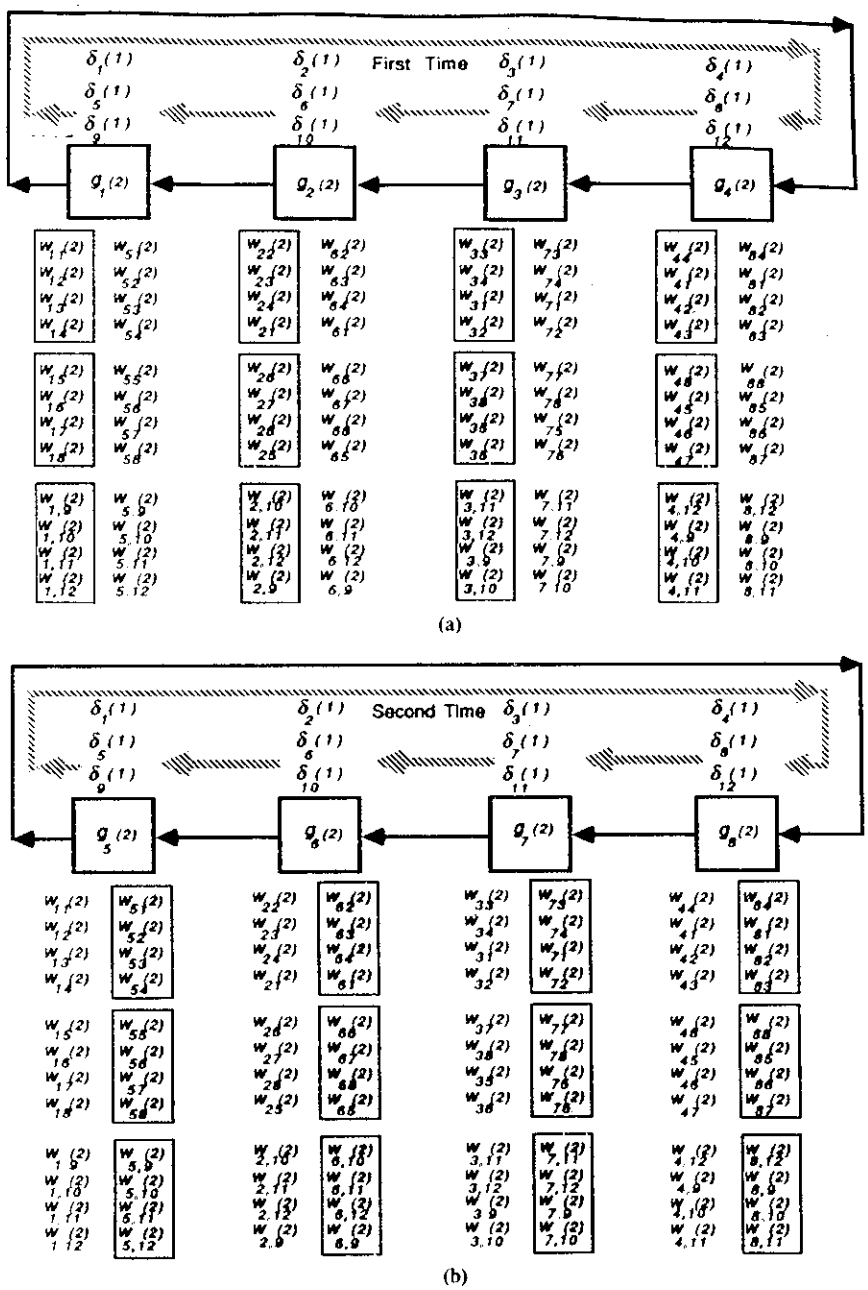


Fig. 12. The ring architecture for the implementation of the consecutive VMM operations of a two-layer perceptron. (a) The accumulation of the first half the back-propagated corrective signals. (b) The accumulation of the second half the back-propagated corrective signals.

[39]. It is also important to compare the neural networks approach with the other conventional methods [44], e.g., simulated annealing [33], hidden Markov model [29], [60], pattern recognition [54], [57], and nonlinear programming [13].

References

[1] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for Boltzmann machines," *Cognitive Sci.*, vol. 9, pp. 147–169, 1985.

[2] H. M. Ahmed, "Alternative arithmetic unit architectures for VLSI digital signal processors," in *VLSI and Modern Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1985, ch. 16, pp. 277–306.

[3] J. Albus, "A new approach to manipulator control: The cerebellar model articulation controller (CMAC)," *Trans. ASME, J. Dynamic Syst., Meas., Contr.*, vol. 97, pp. 220–227, Sept. 1975.

[4] J. Alspector and R. B. Allen, "A neuromorphic VLSI learning systems," in P. Losleben, Ed., *Advanced Research on VLSI*. Cambridge, MA: MIT Press, 1987, pp. 313–349.

[5] S. I. Amari, "Characteristics of randomly connected threshold-element network systems," *Proc. IEEE*, vol. 59, pp. 35–47, Jan. 1971.

[6] J. A. Anderson, *Neurocomputing—Paper Collections*. Cambridge, MA: MIT Press, 1988.

[7] ——, "A simple neural network generating an interactive memory," *Math. Biosci.*, vol. 14, pp. 197–220, 1972.

[8] L. E. Atlas, T. Homma, and R. J. Marks II, "A neural network model for vowel classification," in *Proc. IEEE, ICASSP'87* (Dallas, TX, 1987).

[9] D. H. Ballard and C. M. Brown, *Computer Vision*. Englewood Cliffs, NJ: Prentice Hall, 1982.

[10] C. R. Baugh and B. A. Wolley, "A two's complement parallel array multiplication algorithm," *IEEE Trans. Comput.*, vol. C-22, pp. 1045–1047, Dec. 1973.

[11] D. J. Burr, "Experiments with a connectionist text reader," in *Proc. IEEE, 1st Int. Conf. on Neural Networks* (San Diego, CA, 1987), pp. IV 717–IV 724.

[12] G. A. Carpenter and S. Grossberg, "ART2: Self-organization of stable category recognition codes for analog input patterns," in *Proc. IEEE ICNN'87* (San Diego, CA, 1987), pp. II 727–II 736.

[13] L. O. Chua and G. N. Lin, "Nonlinear programming without computation," *IEEE Trans Circuits Syst.*, vol. CAS-31, pp. 182–188, Feb. 1984.

[14] M. A. Cohen and S. Grossberg, "Absolute stability of global pattern formation and parallel memory storage by competitive neural networks," *IEEE Trans. Syst. Man, Cybern.*, vol. SMC-13, no. 5, pp. 815–825, Sept. 1983.

[15] J. D. Cowan and D. H. Sharp, "Neural nets," Tech. Rep. Math. Dept., Univ. of Chicago, 1987.

[16] R. Eckmiller and C. V. D. Malsburg, *Neural Computers* (NATO ASI Series F, Computer and System Science. New york, NY: Springer-Verlag, 1987.

[17] N. H. Farhat, D. Psaltis, A. Prata, and E. Peak, "Optical implementation of the Hopfield model," *Appl. Opt.*, vol. 24, pp. 1469–1475, May 1985.

[18] K. Fukushima, "Cognitron: A self-organizing multilayered neural network," *Biol. Cybern.*, vol. 20, pp. 121–136, 1975.

[19] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biol. Cybern.*, vol. 36, pp. 193–202, Apr. 1980.

[20] R. P. Gorman and T. J. Sejnowski, "Learned classification of sonar targets using a massively parallel network," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 36, pp. 1135–1140, July 1988.

[21] S. Grossberg, "Adaptive pattern classification and universal recoding: Part I. Parallel development and coding of neural feature detectors," *Biol. Cybern.*, vol. 23, pp. 121–134, 1976.

[22] D. O. Hebb, *The Organization of Behavior*. New York, NY: Wiley, 1949.

[23] G. E. Hinton, "Connectionist learning procedure," Tech. Rep. CMU-CS-87-115, Carnegie Mellon Univ., Pittsburgh, PA: Sept. 1987.

[24] G. E. Hinton and T. J. Sejnowski, "Learning and relearning in Boltzmann machine," in *Parallel Distributed Processing (PDP): Exploration in the Microstructure of Cognition*, vol. 1, Cambridge, MA: MIT Press, 1986, ch. 7, pp. 282–317.

[25] J. J. Hopfield, "Neural network and physical systems with emergent collective computational abilities," *Proc. Nat. Acad. Sci. USA*, vol. 79, pp. 2554–2558, 1982.

[26] ——, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proc. Nat. Acad. Sci. USA*, vol. 81, pp. 3088–3092, 1984.

[27] J. J. Hopfield and D. W. Tank, "Neural computation of decision in optimization problems," *Biol. Cybern.*, vol. 52, pp. 141–152, 1985.

[28] J. N. Hwang, "Algorithms/applications/architecture of artificial neural nets," Ph.D. dissertation, Dept. of Elect. Eng., Univ. of Southern California, Dec. 1988.

[29] J. N. Hwang, J. A. Vlontzos, and S. Y. Kung, "A neural network architecture for hidden Markov models," to appear in *IEEE Trans. Acoust., Speech, Signal Process.*, Dec. 1989.

[30] C. Jorgensen, W. Hamel, and C. Weisbin, "Autonomous robot navigation," *Byte*, pp. 223–235, Jan. 1986.

[31] C. C. Jorgensen, "Neural network representation of sensor graphs for autonomous robot navigation," in *Proc. IEEE, 1st Int. Conf. on Neural Networks* (San Diego, CA, June 1987), pp. IV 507–IV 515.

[32] M. Kawato, Y. Uno, and M. Isobe, "A hierarchical model for voluntary movement and its application to robotics," in *Proc. IEEE, 1st Int Conf. on Neural Networks* (San Diego, CA, June 1987), pp. IV573–IV582.

[33] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecci, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 1983.

[34] C. Koch, J. Marroquin, and A. Yuille, "Analog neuronal networks in early vision," *Proc. Nat. Acad. Sci.*, vol. 83, pp. 4263–4267, 1986.

[35] T. Kohonen, "Correlation matrix memories," *IEEE Trans. Comput.*, vol. C-21, 1972.

[36] T. Kohonen, *Self-Organization and Associative Memory, Series in Information Science*, vol. 8. New York, NY: Springer-Verlag, 1984.

[37] ——, "Self-organized formation of topologically correct feature map," *Biol. Cybern.*, vol. 43, pp. 59–69, 1982.

[38] S. Y. Kung, *VLSI Array Processors*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[39] S. Y. Kung and J. N. Hwang, "An algebraic projection analysis for optimal hidden units size and learning rate in back-propagation learning," in *IEEE, Int. Conf. on Neural Networks, ICNN'88* (San Diego, CA, July 1988), vol. 1, pp. 363–370.

[40] ——, "Parallel architectures for artificial neural nets," in *IEEE Int. Conf. on Neural Networks, ICNN'88* (San Diego, CA, July 1988), vol. 2, pp. 165–172.

[41] ——, "A unified systolic architecture for artificial neural networks," *J. Parallel Distributed Comput.*, (Special Issue on Neural Networks), vol. 6, pp. 358–387, Apr. 1989.

[42] M. Kuperstein, "Adaptive visual-motor coordination in multijoint robots using a parallel architecture," in *IEEE Int. Conf. on Automatic Robotics*, pp. 1595–1602, Mar. 1987.

[43] A. Lapedes and R. Farber, "Nonlinear signal processing using neural networks," in *Proc. IEEE, Conf. on Neural Information Processing Systems—Natural and Synthetic* (Denver, CO, Nov. 1987).

[44] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, vol. 4, pp. 4–22, 1987.

[45] D. G. Luenberger, *Linear and Nonlinear Programming*. Reading, MA: Addison-Wesley, 1984.

[46] J. Lupo, *DARPA: Neural Network Study*. Burke, VA: AFCEA Int. Press, 1988.

[47] W. D. Mao and S. Y. Kung, "Implementation and performance issues of back-propagation neural nets," Tech. Rep. Elec. Eng. Dept., Princeton Univ., Jan. 1989.

[48] D. Marr, "A theory of cerebellar cortex," *J. Physiol. Lond.*, p. 202, 1969.

[49] D. Marr and T. Poggio, "Cooperative computation of stereo disparity," *Science*, vol. 194, pp. 283–287, 1976.

[50] J. L. McClelland and D. E. Rumelhart, "Distributed memory and the representation of general and specific information," *J. Exper. Psychol.: General*, vol. 114, pp. 158–188, 1985.

[51] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in the nervous activity," *Bull. Math. Biophys.*, vol. 5, no. 115, 1943.

[52] W. T. Miller, "Sensor-based control of robotic manipulator using a general learning algorithm," *IEEE J. Robotics Automat.*, vol. RA-3, pp. 157–165, Apr. 1987.

[53] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA: MIT Press, 1969.

[54] N. J. Nilsson, *Learning Machines*. New York, NY: McGraw-Hill, 1965.

[55] D. Parker, "Learning logic," Tech. Rep. TR-47, Center for Computational Research in Economics and Management Science, MIT, Cambridge, 1985.

[56] B. K. Parsiand B. K. Parsi, "Evaluation of network architectures on test learning tasks," in *Proc. IEEE, 1st Int. Conf. on Neural Networks* (San Diego, CA, June 1987), pp. III785–III790.

[57] T. Pavlidis, *Structural Pattern Recognition*. New York, NY: Springer-Verlag, 1977.

[58] D. Psaltis, A. Sideris, and A. Yamamura, "A hierarchical model for voluntary movement and its application to robotics," in *Proc. IEEE, 1st Int. Conf. on Neural Networks* (San Diego, CA, June 1987), pp. IV551–IV558.

[59] D. Psaltis, K. Wagner, and D. Brady, "Learning in optical neural computers," in *Proc. IEEE, 1st Int. Conf. on Neural Networks* (San Diego, CA, June 1987).

[60] L. R. Rabiner and B. H. Juang, "An introduction to hidden Markov models," *IEEE ASSP Mag.*, vol. 3, no. 1, pp. 4–16, Jan. 1986.

[61] R. Rastogi, P. K. Gupta, and R. Kumaresen, "Array signal processing with interconnected neuron-like elements," in *Proc. IEEE ICASSP'87* (Dallas, TX), pp. 54.8.1–54.8.4, 1987.

[62] G. N. Reeke and G. M. Edelman, "Selective neural networks and their implications for recognition automata," *Int. J. Supercomput. Appl.*, pp. 44–69, 1987.

[63] F. Rosenblatt, "The perceptron: A probablistic model for information storage and organization in the brain," *Psych. Rev.*, vol. 65, 1958.

[64] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing (PDP): Exploration in the Microstructure of Cognition*, vol. 1. Cambridge, MA: MIT Press, 1986, ch. 8, pp. 318–362.

[65] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, *Parallel Distributed Processing (PDP): Exploration in the Microstructure of Cognition*, vol. 1. Cambridge, MA: MIT Press, 1986.

[66] D. E. Rumelhart and D. Zipser, "Feature discovery by competitive

learning, *Cogn. Sci.*, vol. 9, pp. 75-112, 1985.
[67] T. J. Sejnowski, "Parallel networks that learn to pronounce English text," *Complex Syst.*, vol. 1, pp. 145-168, 1987.
[68] K. G. Shin and S. B. Malin, "A structured framework for the control of industrial manipulator," *IEEE Trans. Syst. Man, Cybern.*, vol. SMC-15, pp. 78-94, Jan./Feb. 1985.
[69] K. Steinbush, "The learning matrix, *Kybernetik* (*Biol. Cybern.*), pp. 36-45, 1961.
[70] C. V. Stewart and C. R. Dyer, "Neural network representation of sensor graphs for autonomous robot navigation," in *Proc. IEEE, 1st Int. Conf. on Neural Networks* (San Diego, CA, June 1987), pp. IV215-IV224.
[71] G. Z. Sun, H. H. Chen, and Y. C. Lee, "Neural network representation of sensor graphs for autonomous robot navigation," in *Proc. IEEE, 1st Int. Conf. on Neural Networks* (San Diego, CA, June 1987), pp. IV345-IV356.
[72] M. Takeda and J. W. Goodman, "Neural networks for computation: Number representations and programming complexity," *Appl. Opt.*, vol. 25, pp. 3033-3046, Sept. 1986.
[73] D. W. Tank and J. J. Hopfield, "Simple "neural" optimization networks: An A/D converter, signal decision circuit, and a linear programming circuit," *IEEE Trans. Circuits Syst.*, vol. CAS-33, pp. 533-541, 1986.
[74] C. E. Thorp, "Path relaxation: Path planning for a mobile robot," in *Proc. of the AAAI* (Austin, TX, Aug. 1984), pp. 318-321.
[75] K. Tsutsumi and H. Matsumoto, "Neural network representation of sensor graphs for autonomous robot navigation," in *Proc. IEEE 1st Conf. on Neural Networks* (San Diego, CA, June 1987), pp. IV525-IV534.
[76] P. J. Werbos, "Beyond regrssion: New tools for prediction and analysis in the behavior science," Ph.D. dissertation, Harvard University, Cambridge, MA, 1974.
[77] B. Widrow and R. Winter, "Neural nets for adaptive filtering and adaptive pattern recognition," *IEEE Comput. Mag.*, vol. 21, pp. 25-39, Mar. 1988.
[78] G. Widrow and M. E. Hoff, "Adaptive switching circuit," in *IRE Western Electronic Show and Convention: Convention Record*, pp. 96-104, 1960.
[79] D. J. Willshaw, "Models of distributed associative memory models," Ph.D. dissertation, University of Edinburgh, Scotland, 1971.
[80] Y. T. Zhou and R. Chellappa, "Stereo matching using a neural network," in *Proc. IEEE ICASSP'88* (New York, NY, Apr. 1988), pp. 940-943.

Sun-Yuan Kung (S'75-M'78-SM'84-F'88) received the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA in 1977.

From 1974 to 1977, he was associated with the Amdahl Corporation, Sunnyvale, CA, as an Associate Engineer in LSI design and simulation. In 1977, he joined the faculty of Electrical Engineering—Systems at the University of Southern California, Los Angeles, where he became a Professor in 1986. During 1984, he was a Visiting Professor at Stanford University, Stanford, CA, and a Visiting Professor at the Delft University of Technology, Delft, The Netherlands. In 1987 he joined the faculty of the Department of Electrical Engineering at Princeton University, Princeton, NJ, as a Professor. His research interests include linear system approximation, modern spectrum analysis, digital signal processing, and VLSI array processors. He has authored a texbook *VLSI Array Processors* (Englewood Cliffs, NJ: Prentice-Hall) and over 150 technical papers, and was Keynote Speaker and Program Chairman, respectively, for the 1986 and 1988 International Conference on Systolic Arrays.

From 1984 to 1985 Dr. Kung was the Associate Editor for the VLSI area of the IEEE TRANSACTIONS ON ACOUSTIC, SPEECH, AND SIGNAL PROCESSING and is currently a member of the IEEE ASSP Administration Committee and Technical Committee on VLSI. He served as the General Chairman of the IEEE workshops on VLSI Signal Processing in 1982 and 1986.

Jeng-Neng Hwang (M'89) received the B.S. and M.S. degrees, both in electrical engineering, from the National Taiwan University, Taipei, Taiwan, and the Ph.D. degree in electrical engineering from the University of Southern California, Los Angeles, in December 1988.

Since August 1989, he has been with the Department of Electrical Egineering, University of Washington at Seattle, as an Assistant Professor. His research interests include parallel algorithm design, VLSI array architecture, neural networks, and signal/image processing.

Dr. Hwang is a member of ACM.