USC-SIPI REPORT #141

Algorithms/Applications/Architectures of Artificial Neural Nets

by

Jenq-Neng Hwang

December 1988

Signal and Image Processing Institute
UNIVERSITY OF SOUTHERN CALIFORNIA
Department of Electrical Engineering-Systems
Powell Hall of Engineering
University Park/MC-0272
Los Angeles, CA 90089 U.S.A.

Acknowledgements

I wish to express my sincere thanks and appreciation to Professor S. Y. Kung, my research supervisor, for his constant guidance and encouragement during the three years of my Ph.D. studies in the Department of Electrical Engineering at the University of Southern California. Prof. Kung has been an advisor and a dear friend, whose help to me is invaluable in many ways. I also wish to thank Prof. R. Leahy and Prof. M. D. Huang for their advice and for their helpful comments on this dissertation.

Thanks are extended to the many graduate students I have worked with. I would like to especially thank Dr. P. S. Lewis for his friendship, our valuable discussions, and his technical editing on this dissertation. I would also like to thank Dr. S. C. Lo, Dr. S. N. Jean, Mr. J. A. Vlontzos, and Mr. S. W. Sun for their very helpful collaboration and contributions to the research performed in this dissertation. My good friend Mr. J. C. Lien deserves my special thanks for helping me deal with all of the necessary document processing while I was visiting at Princeton University. I also wish to express my gratitude to Mrs. Linda Varilla of the Signal and Image Processing Institute at USC, whose professional help was invaluable to me during the past three years.

I gratefully acknowledge the financial support provided by the National Science Foundation under Grant ECS-82-13358, and by the Innovative Science and Technology Office of the Strategic Defense Initiative Organization, administered through the Office of Naval Research under Contract No. N00014-85-K-0469 and N00014-85-K-0599.

Finally, I would like to dedicate this work to my parents and my wife, Ming-Ying, for their support, help, and tolerance during the past three years, and my new born daughter, Jaimie, for all the joys and cheers she brought into this family.

Contents

List of Figures

Abstract

In the research field of artificial neural nets (ANNs), important contributions have come from cognitive science, neurobiology, physics, computer science, control, analog VLSI, and optics. This dissertation is intended to broaden the present scope so as to achieve an in-depth understanding of neural computing from the perspectives of parallel algorithm design, digital VLSI array architectures, digital signal processing, and numerical analysis. Therefore, in a broader sense, the aim of this dissertation has been to accomplish truly cross-disciplinary research.

A variety of neural nets are considered, including single layer feedback neural nets (e.g., Hopfield neural nets, Rumelhart memory/learning modules, and Boltzmann machines), multilayer feed-forward nets (e.g., multilayer Perceptrons), and hidden Markov models (HMMs). Algorithmic studies based on algebraic projection (AP) analysis are presented to deal with two critical and important issues in back propagation (BP) learning: the selection of the optimal number of hidden units and the optimal learning rate. Potential applications of ANNs to two promising areas, computer vision and robotic processing, are discussed. A systematic algorithm mapping methodology is proposed for mapping neural algorithms into VLSI array architectures. This methodology derives array architectures for the algorithms via a two-stage design, i.e., dependence graph (DG) design and array processor design. Programmable ring systolic ANNs (linear and cascaded) are developed, which can exploit the strength of VLSI and offer intensive and pipelined computing. Both the retrieving and learning phases are integrated in the design. The proposed architectures are more versatile than other existing ANNs; therefore, they can accommodate the most widely used neural nets. A unifying viewpoint is proposed for multilayer Perceptrons and HMMs, and the ring systolic array architectures can further be extended to the application domain of implementing both the scoring and learning phases of HMMs. Finally, some mathematical aspects for ANNs worthy of further pursuit are also presented, which include expressibility and discrimination capabilities, generalization capability, convergence in the retrieving and learning phases, and merging of multilayer Perceptrons and HMMs.

Chapter 1

Introduction

The power of neural nets hinges upon their distinct features of robust processing and adaptive capability in new and noisy environments. It is estimated that the human brain contains over 100 billion (10^{11}) neurons. Neurons possess tree-like structures (called *dendrites*) sp[illegible]alized 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. At least 50 different *neurotransmitter* molecules, which act across synapses, have been identified. These molecules open up neural membrane channels that permit ionic currents to act, enabling the neurons to communicate. Neurotransmitters and currents either depolarize the membrane so that the neuron is excited (i.e., excitatory weights), or else they hyperpolarize it so that the neuron is inhibited (i.e., inhibitory weights) [31,140,141,8].

There is a very high degree of *specificity* in the pattern of inter-neural connectivity. Differing neural types are found in particular regions of the brain, and the neural interconnections between them occur in what appear to be fairly precise patterns. There is also a growing body of evidence which suggests that there is considerable developmental *plasticity* in the neural interconnections in the ability to reorganize and regenerate new patterns of connectivity [31].

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 slower than current computer elements, the brain has a thousand-fold greater connectivity than today's supercomputers [34].

It has been postulated that the primary function of neocortical nets 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. The granule cells in the cerebellum perform the task of decorrelating activity patterns (for efficient storage) arriving along mossy fibers. The resulting patterns are stored in the cerebellum between granule and Purkinje cells via the Hebbian learning rule [31]. Furthermore, because its primary function is memory. the cerebellum has a strikingly regular and simple architecture.

Many examples of biological applications are also available as a useful clue for the development of artificial neural nets (ANNs). 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 lightness 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 net finds a "solution." Interestingly, this process is analogous to the *relaxation* process used in many numerical algorithms run on current digital computers.

1.1 Historical Review of Artificial Neural Nets

Various neural nets models have been proposed with supports of biological evidences during the past 30 years [8,142,147]. Recently, due to the successes of Hopfield's neural circuits [52,53,54,156], Rumelhart's parallel distributed processing models [140,141], and along with matured VLSI and optical technologies, wide surge of interests are brought into the research of artificial neural nets. The history of ANN research can be roughly classified by the evolution of four different types of learning rules: Hebbian learning, delta learning, competitive learning, and probabilistic learning.

Hebbian Learning and Associative Memory One line of evolution is to emulate our human memory, which are not only distributed but also associative—one memory may elicit another if the two have sufficient overlap. This line of evolution starts from the Hebbian rules proposed by Hebb in his book called *The Organization of Behavior* [47]. This book reflected the distributed representation concept, the antithesis of regional localization, of the brain, i.e., many regions of the brain is *equipotent* with respect to a function in spite of the regional specificity (or these regions can implement the same given task). Hebb also proposed that the connectivity of the brain is continually changing as an organism learns differing functional tasks, and cell assemblies are created by such changes. The mechanism that Hebb postulated was that repeated activation of one neuron by another, across a particular synapse, increased its conductance so that groups of weakly connected cells, if synchronously activated, would tend to organize into more strongly connected assemblies. The cell assembly theory has triggered many investigations of learning in neural nets.

The studies of neural nets for associative memory was initiated by *Taylor net* [158,157] and *Learning Matrix* [149,150]. These two nets adopted a Hebbian-like learning rule, called *pre-synaptic facilitation*, with winner-take-all or grandmother cell mechanism so that only a single highly meaningful active cell is detected. Taylor suggested that the association areas of cerebral cortex and thalamus contained such networks. Following Learning matrix, many other workers have independently devised similar associative nets, e.g., Anderson [11,9], Willshaw [174,173], Marr [114], Amari [6,7], and Kohonen [75,74]. These nets are now generally referred to as *associative content addressable memories (ACAM).*

Two researches, Kohonen [75] and Anderson [9], have independently pursued the same model for associative memory based on a correlation matrix. The linear associator model assumed that the input-output relations of the system were specified as a simple matrix-vector multiplication. The learning rule proposed was a generalization of the Hebb synapse, i.e., the synaptic change was proportional to the product of pre- and postsynaptic activities [9], or to the correlation between elements of the input and output vectors [75]. The connection matrix then became the sum of the outer products between the corresponding input and output vectors. By having the same sets of input and output vectors, the system removed an asymmetry between different elements at the synaptic level, which allows the feedback of the outputs to the inputs for another pass through the system. This system, called "autoassociative" by Kohonen, served as the basis for several later feedback network models proposed with strong nonlinearity imposed between the feedback outputs and inputs [10,52]. If the input and the output vectors are different. Kohonen called the system "heteroassociative" (pattern association). The major criticism of this class of models is their linearity, but the useful associative properties, generated by the Hebb synapse, carry over to the later, more complicated, nonlinear systems.

Expanding earlier linear associator research work, Hopfield and his colleagues elegantly made use of the notion of an Liapunov energy function to demonstrate the convergence of an iterative computational model for associative memory and optimization applications [52,53,54,156]. This resulting model, referred to as the Hopfield net, initiated the modern era of the neural nets research. Instead of proposing a learning rule for the neural net, Hopfield tried to show that the function of the nervous system is to develop a number of locally stable points (attractors) in the state space. Since deviations from the stable points disappear [52], this approach provides a mechanism for correcting errors and completing missing information. Making a symmetric assumption about the synaptic weights of the net, Hopfield adopted the slightly modified correlation matrix to construct the weight matrix for the discrete associative memory, this ensures symmetry of modification magnitude for the two allowable output states (0 and 1) of a neuron cell. The dynamic evolution of the system state follows a simple rule and is asynchronous (sequential). This evolution can be regarded as an energy minimization that continues until a stable state (local energy minimum) is reached. Computer simulations and theoretic analysis concluded that the number of "memories" that can be stored was about 15% of the dimensionality. This percentage is similar to that of the correlation matrix based linear associators.

In a later paper [53], Hopfield discussed networks of neurons that exhibit graded intermediate states. The purpose of these states was to imitate the continuous input-output relationship and to simulate the integrative time delay of real neurons. Hopfield showed that the evolution of this continuous state net is also decreasing a Liapunov energy function. Hopfield and Tank [54] built analog circuits based on the continuous state neural net to solve the traveling salesman problem, a difficult NP-complete problem. Although a globally optimal solution is not guaranteed, the analog circuits can reliably find the local minimum of the tour length energy surface nearest to the starting tour. The circuits have also been used to solve some other less complicated problems, e.g., A/D converter and linear programming problems [156].

Kosko further extended Hopfield model to perform hetero-association tasks by introducing bidirectional feedback iterations. The system is called bidirectional associative memory (BAM) [78]. Instead of feedbacking the outputs (through nonlinearity) directly, this system iterate through another set of weights to generate the inputs. The BAM can also be shown to perform the gradient descent search on a Liapunov energy function.

Delta Learning and Pattern Classification The capability of pattern classification or recognition is essential to most animals. McCulloch and Pitts net was the first attempt to explore these capabilities without learning mechanism [118].

Ten years later, Rosenblatt showed that the McCulloch and Pitts net with modifiable connections could be trained (by Perceptron learning rule) to classify certain sets of pattern - this is called Perceptron. Shortly after Rosenblatt's first publications there appeared a closely related variant of the Perceptron, invented by Widrow and Hoff, called *Adaline* (adaptive linear neuron). Although the delta learning rule used in Adaline shares some common properties with Perceptron learning rule (e.g., the error between target and actual output is used), the main difference is that Adaline computes the error before the nonlinear operation, while Perceptron computes the error after the nonlinear operation. There are, however, limits to the performance of Perceptrons and Adaline. Papert and

Minsky proved that the single-layer Perceptron cannot compute simple logical switching functions like XOR without introducing more layers (hidden layers) due to the limited linear separabilities [121]. This is the reason why Madaline (many Adalines) was introduced to overcome this problem [61,123]. Both in the Perceptron and Madaline nets, the basic idea of multiple layers of neurons has been mentioned, however, there was no efficient and successful learning rules proposed. It was not until Werbos [169], Parker [125], and Rumelhart [139] independently proposed the back-propagation (BP) (a generalized delta) learning rule for multilayer Perceptron.

The derivation of the weight updating procedure in back-propagation learning is based on an iterative gradient descent algorithm designed to minimize the total mean squared error between the the desired target values and the actual output values. Since the learning is supervised, the actual outputs are compared with the target values to derive error signals. These error signals are recursively propagated backward from the output layer to the hidden layers, updating of the synaptic weights in all the hidden layers. The NETTalk was the first novel application that a 2-layer Perceptron was trained to read and speak English text using back-propagation [144].

Competitive Learning and Feature Detection The competitive learning was introduced to perform the regularity extraction, in which the system develop its own feature representations of the input training patterns without a priori category assignment. The learning rule is similar to the clustering techniques used in the vector quantization approach. It iteratively classifies the input training patterns into a set of clusters to update the centroid information, and those converged centroid information are memorized in the synaptic weights for the future match filtering applications. The neural net architecture for this learning has a basic layer structure, connections between layers are excitatory, and within layers are inhibitory. Different manipulations of the weight changes lead to different models. Popular ones are adaptive resonance theory (ART) [44,26], Cognitron/Neocognitron [38,39], self-organized feature map [77,76].

The ART network implements a clustering algorithm which is very similar to the simple sequential leader clustering algorithm [109]. The leader algorithm selects the first input as the exemplar for the first cluster. The next input is compared to the first cluster exemplar. It follows the leader and is clustered with the first if the distance to the first is less than a threshold (vigilance test). Otherwise it is the exemplar for a new cluster. This process is repeated for all the following inputs.

The brain is organized in many places so that aspects of the sensory environment are represented in the form of two-dimensional maps. The feature map network is attempting to construct an artificial system that can show the same behavior. This can be achieved by responding nearby neurons similarly. Based on biological evidences, Kohonen constructed the feature map system where nearby neurons excite each other, with inhibition for longer distances. The size of the excitatory region is changed during learning, starting out large and shrinking as organization proceeds.

Unlike the back-propagation learning, where the target function is specified only in the output layer, and the weight updatings of hidden layers are performed by recursively propagating the error backward. The multilayer learning system Neocognitron proposed in [39] assumed that the teacher knows "roughly" what features will be required and that learning progresses sequentially from the input layer to the output layer.

Probabilistic Learning and Global Optimization The final class of learning rule is the probabilistic learning. Except the hidden Markov model, which will be discussed in more details, most other neural nets in this class are motivated by simulated annealing technique to search the global optimum solution.

At about the same time when Hopfield showed how parallel networks can be used to access memories that were stored as local minimum, Kirkpatrick introduced a p..ysical analogy technique, which allows energy surface smoothing and occasional uphill climbing to search for the global minimum. This is so called *simulated annealing* [72].

The simulated annealing was then used effectively for image restoration by Geman and Geman [40], who also established limits on the allowable speed of the annealing schedule. Hinton and his colleagues [14,1] applied the simulated annealing approach to the Hopfield net, the resulting nets is called *Boltzmann machine*. Instead of using uniform probability density for generating the state transition as in Boltzmann machine, the Cauchy machine adopted the Cauchy density function, which will lead to a faster annealing schedule [153].

1.2 Basic ANN Model

Neuroscientists have revealed that the processing power in the human brain lies in a large number of identical *processing units* or *neurons*, linked to each other with variable strengths in terms of a network of *synaptic weights*. In order to provide a certain degree of biological fidelity, all of the existing ANN models adopt an information storage/retrieval process which involves *altering* the pattern of connections among a large number of primitive cells, and/or by *modifying* certain weighting parameters associated with each connection [81].

A basic ANN model comprises a *propagation rule* and a *nonlinear activation* as shown in Figure 1.1. Each neural processing unit (PU) has an activation value $a_i(k)$ at (time or spatial) index k. This value (either discrete or continuous) is propagated through a network of unidirectional connections to other PUs in the system. Associated with each connection, there is a *synaptic weight* denoted as w_{ij} which dictates the effect the j-th PU has on the i-th PU. All of the inputs to the i-th PU from other PUs are accumulated, together with the external input θ_i to yield the net input $u_i(k+1)$ according to the following *propagation rule*:

$$u_i(k+1) = \sum_j w_{ij} a_j(k) + \theta_i \tag{1.1}$$

The net input value $u_i(k+1)$, along with its current activation value $a_i(k)$ and last net input value $u_i(k)$, will determine the new activation value $a_i(k+1)$ by the *nonlinear activation* function f:

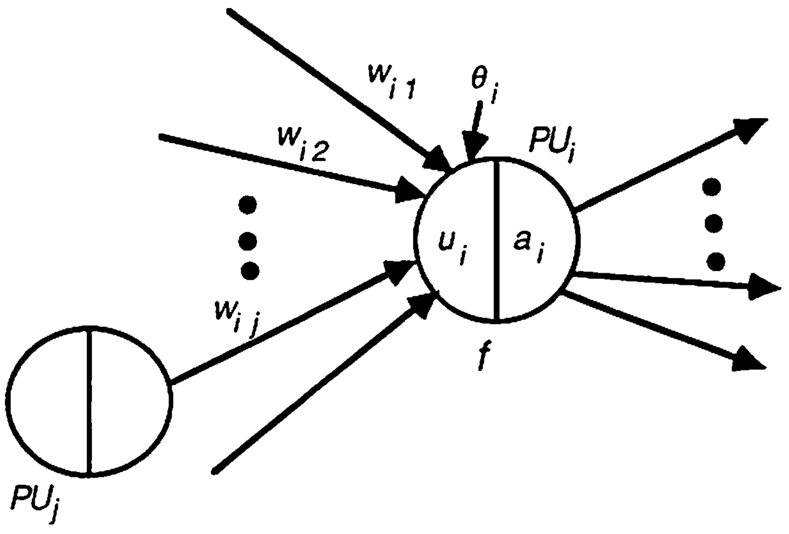


Figure 1.1: A basic ANN model with two operations: propagation rule, and nonlinear activation.

$$a_i(k+1) = f(u_i(k+1), u_i(k), a_i(k)) \tag{1.2}$$

The activation function f can be a deterministic or stochastic function. For example, three popular deterministic activation functions are: step, sigmoid, and squashing functions [140,141].

1.3 Three Aspects of ANN Research

The three most important aspects of ANN research are: ***algorithms, applications***, and ***architectures***. Each aspect will be elaborated on in later chapters, here a brief introduction is provided:

1.3.1 Algorithms

The algorithms for neural net processing can be divided into two phases: *retrieving* and *learning*.

Retrieving Phase

Various kinds of (nonlinear) system dynamics have been proposed for the retrieving phase of an ANN. Basically, the neuron, say i-th neuron, updates its old activation value $a_i(k)$ based on the system dynamics equations given in Eq. 1.1 and Eq. 1.2 to form the new activation value $a_i(k+1)$, i.e.,

$$\begin{aligned} u_i(k+1) &= \sum_j w_{ij} a_j(k) + \theta_i \\ a_i(k+1) &= f(u_i(k+1), u_i(k), a_i(k)) \end{aligned} \tag{1.3}$$

and the indices k will be continuously incremented until a stop criterion is reached.

Learning Phase

A salient feature of a neural net is its capability of adaptation or self-organization, i.e., adaptive updating of the synaptic weights by specific "learning" rules [50]. The weight updating for incorporating the knowledge in the m-th presentation of training pattern (or patterns) can be performed by;[1]

[1]To speed up the convergence and avoid leading the system to oscillation, the momentum factor can be considered [139,126]

$$w_{ij}^{(m+1)} = w_{ij}^{(m)} + \Delta w_{ij}^{(m)} + \beta\,(w_{ij}^{(m)} - w_{ij}^{(m-1)})$$

where the momentum factor β is an exponential learning decay factor between 0 and 1.

$$w_{ij}^{(m+1)} = w_{ij}^{(m)} + \Delta w_{ij}^{(m)} \tag{1.4}$$

To handle different applications, there are two categories of learning algorithms. In supervised learning algorithms a desired output response must be specified (e.g., delta learning rule [172], back-propagation learning rule [139]); whereas in unsupervised learning algorithms no desired response is necessary (e.g., Hebbian learning rule [47], competitive learning rule [44]).

Take, for example, one popular supervised learning algorithm, the delta learning rule (or Widrow-Hoff rule), where a target value vector $\{t_i\}$ is specified for every set of input vector, and the adjustment is proportional to the difference between the target values $\{t_i\}$ and the actual output values $\{a_i\}$, i.e.,

$$\Delta\, w_{ij} = \eta\; (t_i - a_i)\; a_j \tag{1.5}$$

where η is the learning rate which regulates the rate of adjustment of the weights.

1.3.2 Applications

Successfully neural net applications may be grouped into two classes: *optimization* and *associative retrieval/classification*.

Optimization

For optimization applications, neural nets are used as a state space search mechanism. The (fixed) synaptic weights are set before the retrieving (search) process is started. There is no explicit learning procedure for setting the weights, the derivation of $\{w_{ij}\}$ usually hinges upon finding a Liapunov energy function for the specific application.[2] For example, if the Hopfield neural net is used for solving an optimization problem, the Liapunov energy function has the following form [53,54]:

$$E = -\frac{1}{2}\sum_i \sum_j w_{ij} a_i a_j - \sum_i \theta_i a_i \tag{1.6}$$

In some applications, the Liapunov energy function is directly available (e.g., regularized least squares applications [81]). 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 [54]). Once the synaptic weights are determined from the energy function, then the retrieving phase can be performed by following the system dynamics in Equations 1.1 and 1.2. Basically, the iterations execute a gradient descent search 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 net, but uses the simulated annealing technique [51]. Simulated annealing is a search technique that make it possible to escape from the trap of a local optimum. The mechanism can be viewed as a flattening of the trap along with a possibility, based on a stochastic decision, of accepting an updating which (temporally) corresponds to a *worse* (higher energy) solution [72,153]. It has been proved that, with a proper

[2]A Liapunov energy function is monotonically decreasing along a time evolution, i.e., $\frac{d}{dt}E \leq 0$.

annealing schedule [40], the states of the Boltzmann machine will gradually move toward the global optimal solution.

Associative Retrieval, Classification, and Generalization

Another very promising application of neural processing is that of associative retrieval, classification, and generalization. The associative retrieval problem 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 classification problem is to identify the corresponding category for any test pattern. The retrieving phase uses the same system dynamics (see Eqs. 1.1 and 1.2) as in the optimization applications.

The generalization problem is discover statistically salient features of the input population, i.e., detect the regularity. Unlike the classification, there is no *a priori* set of categories into which the patterns are to be classified; rather, the network must develop its own featural representation of the input training patterns. For this application class, learning schemes are often adopted to train the synaptic weights.

Applications in Computer Vision and Robotic Processing

To successfully address a particular application requires understanding of the signal formation process, the algorithm class involved, and the specifics of the system. The immense computational complexity inherent in mimicking the human visual perception process, or human dynamic movement control, require massive amounts of processing power. On the other hand, humans perform these tasks easily by using neural nets. It becomes obvious that two potential applications suitable for ANNs are computer vision and the robotic processing. Most of algorithms in these two application domains can be either formulated as optimization (or constrained optimization) problems or pattern association tasks, thus they can be efficiently implemented by ANNs.

1.3.3 Architectures

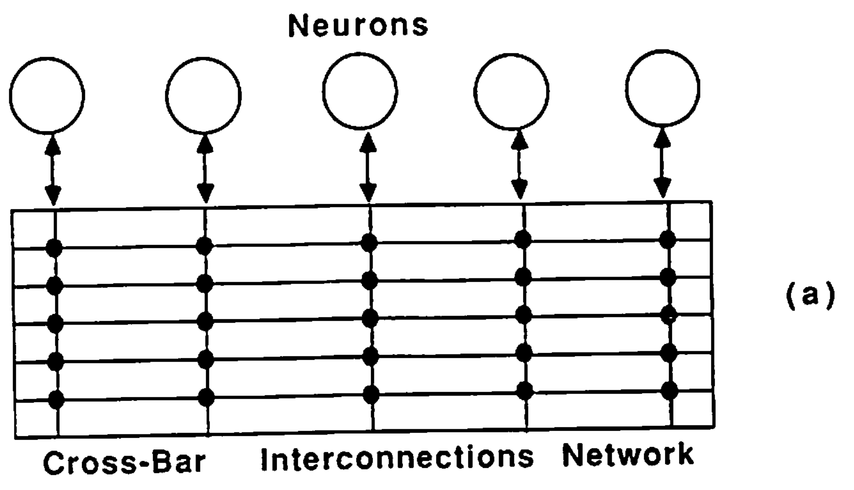
By taking advantage of now mature algorithm mapping methodologies [81] and CAD technology [119]. VLSI arrays can be systematically derived from the dependency structure of neural net algorithms. This dissertation proposes a highly pipelined systolic/wavefront array design [80,81] for implementing ANNs. These arrays take advantage of VLSI's strength in intensive and pipelined computing, yet circumvent its main limitation on communication. The proposed design can offer a greater flexibility than the existing neural net architectures, since it provides a general-purpose programmable array architecture for both optimization and associative retrieval applications. The functional complexity of the processor elements (PEs) and the interconnection structure of the array system are influenced respectively by the *neural processing units* and the *connectivity pattern*.

An important consideration in the architectural design is to ensure that the processing in both the learning phase and the retrieving phase can share the same storage and/or processing hardware. This will not only speed up real-time learning, but also avoid the difficulty of reloading of synaptic weights for retrieving. Another design consideration is to find a proper digital arithmetic technique (such as CORDIC) to efficiently compute the propagation rule and the nonlinear activation function.

The overall ANN array architecture is largely dictated by the connectivity patterns involved. The neural net can be a single-layer (module) feedback network or a multilayer feed-forward network. In a single-layer feedback neural net, the neural processing units are interconnected through synaptic weights network as shown in Figure 1.2(a). Each neuron unit receives the feedback inputs from the other neuron units in the same layer as well as external inputs from external sources (e.g., other networks or modules). It is also possible to cascade several single-layer feedback neural nets to form a larger net [117,107]. In a feed-forward multi-layer neural net, there are one or more layers of *hidden neuron units* between the input and output neuron layers. The neurons of a hidden layer receive the inputs from the the neurons at the lower layer (or layers) and send the outputs to the neurons at the upper layer (or layers) as shown in Figure 1.2(b). Inhibitions within the same layer are also frequently incorporated.

1.4 Overview of the Dissertation

In the research field of artificial neural nets (ANNs), important contributions have come from cognitive science, neurobiology, physics, computer science, control, analog VLSI, and optics [8,142,147,120]. This dissertation is intended to



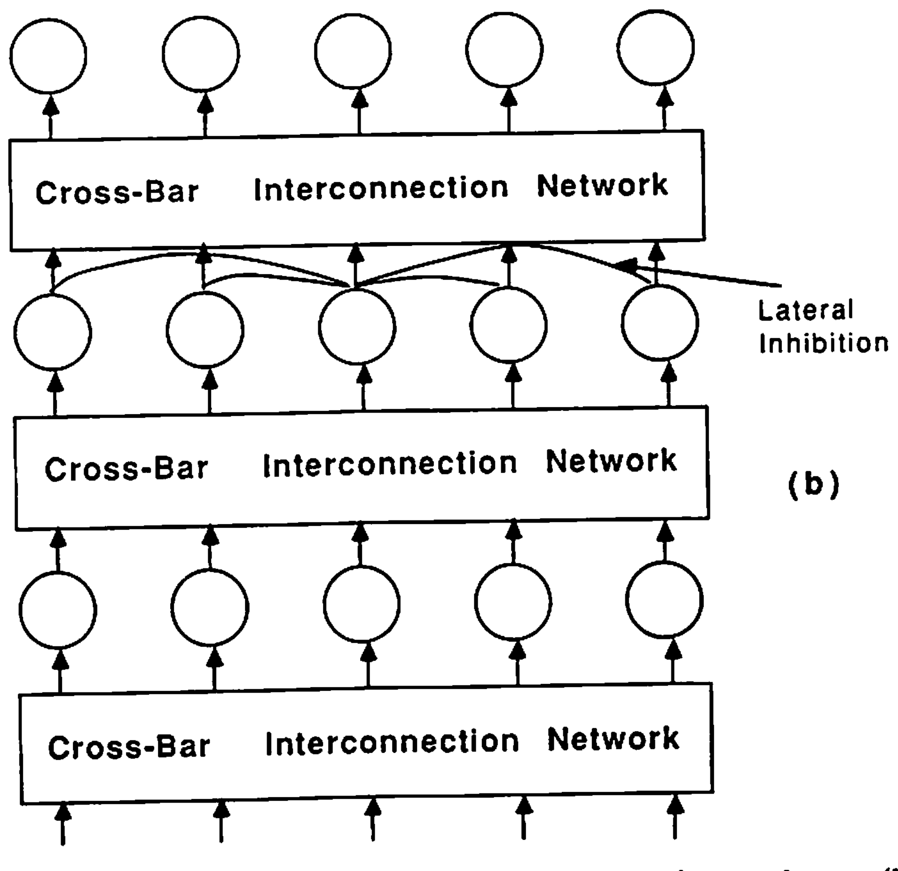

Cross-Bar Interconnection Network
Lateral Inhibition
Cross-Bar Interconnection Network
(b)
Cross-Bar Interconnection Network

Figure 1.2: (a) A fully interconnected single-layer feedback neural net. (b) A fully interconnected multilayer feed-forward neural net.

broaden the present scope so as to achieve an in-depth understanding of neural computing from the perspectives of parallel algorithm design, digital VLSI array architectures, digital signal processing, and numerical analysis. Vertically integrated array architecture design (see Figure 1.3) depends on a fundamental understanding of algorithms, applications, and architectures. Therefore, array processor design involves a very broad spectrum of disciplines, including algorithm analyses, parallelism extraction, array architectures, programming techniques, functional primitives, structural primitives, and numerical performance of algorithms. Thus, in a broader sense, this dissertation is aimed at accomplishing truly cross-disciplinary research.

Chapter 2 starts from a brief introduction of various widely used retrieving and learning algorithms for ANNs. For example, single-layer feedback neural nets (e.g., the Hopfield neural nets [52,53,54,15[illegible]], the Rumelhart memory/learning modules [117], and the Boltzmann machine [1,51]), and feed-forward multilayer Perceptrons [138,121,139,109].

Chapter 3 deals with two critical and important issues in back propagation learning: the selection of the optimal size of hidden units and the optimal learning rate [92]. The number of hidden units must be sufficient to provide the discriminating capability required by the given application [123], but an excessively large number of synaptic weights may lead to costly and unreliable training. Therefore, it is very desirable to have an *a priori* estimate of the optimal number of hidden neurons. It is also important to determine the optimal learning rate to permit fast learning without incurring instability of the iterative computation.

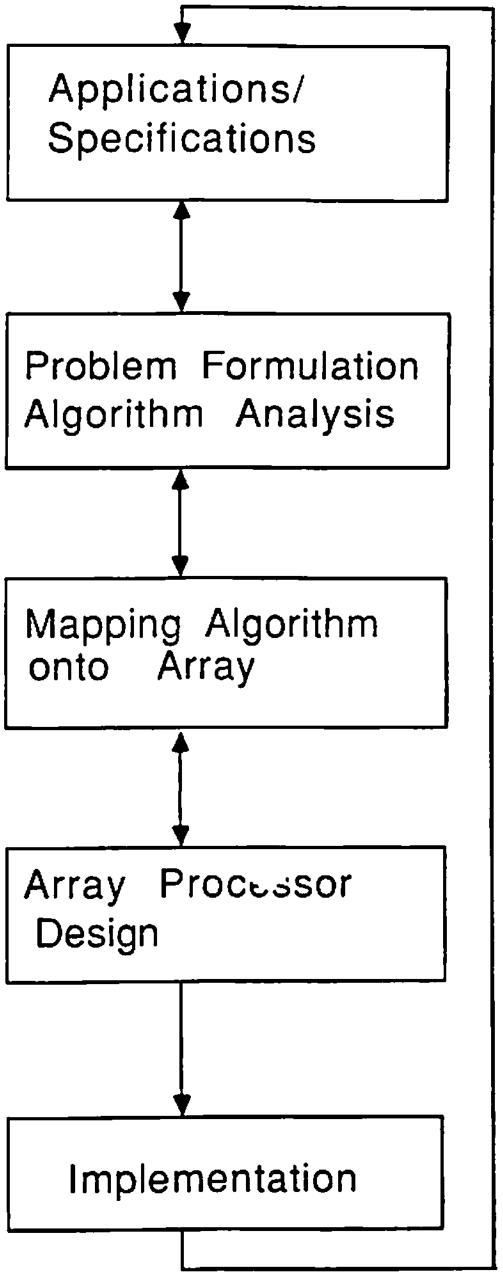

Applications/ Specifications
Problem Formulation Algorithm Analysis
Mapping Algorithm onto Array
Array Processor Design
Implementation

Figure 1.3: Vertically integrated array architecture design.

Chapter 4 discusses the two most promising application domains of ANNs: computer vision and robotic processing. Without loss of generality, typical examples are given for demonstrating the usefulness of ANNs: e.g., the stereo matching problem in computer vision, and optimal path planning and dynamic movement control in the robotic processing [95].

Chapter 5 advocates a programmable ring systolic ANN, which uses the strengths of VLSI in terms of intensive and pipelined computing and circumvents the limitations of VLSI in communication. A systematic mapping methodology is first introduced via a two stage design. This methodology is used to derive the array architectures for the neural net algorithms. This systolic ANN is meant to be more general purpose than most other ANN architectures proposed. It may be easily adapted to a variety of different algorithms in both the retrieve and learning phases of ANNs. Hardware implementation for the processing units based on CORDIC techniques is also discussed.

Chapter 6 further extends the application domain of the systolic ANN to implementation of both the scoring and learning phases of hidden Markov models (HMMs). A unifying viewpoint between HMMs and multilayer Perceptrons is first presented. The array architecture can thus be naturally derived. By appropriately scheduling the algorithm, which combines both the operations of the backward evaluation and the reestimation procedures, we can use this systolic array for HMM in a most efficient manner. This ring systolic array for HMM can also be easily adapted to execute the left-to-right HMMs with significant time saving, by using bidirectional semi-global links. With little extra effort in the computations of forward and backward likelihood variables, this architecture can also incorporate the scaling scheme used to prevent mathematical underflow problems.

In the final chapter. Chapter 7, potential future research directions worthy of further investigation are discussed. These include: expressibility and discrimination capabilities: generalization capability; convergence in the retrieving and learning phases; and merging of multilayer Perceptrons and HMMs.

1.5 Original Contributions

This dissertation presents unifying research work in the algorithmic, applicational, and architectural studies of ANNs. The significant contributions of this work are listed in the following.

1. An algebraic projection analysis is proposed to study the dynamic behavior of the back-propagation (BP) learning algorithm. The analysis successfully predicts the optimal hidden units and learning rate for efficient training of the multilayer Perceptron [101,92].

2. A thorough study of two potential application domains of ANNs, computer vision and robotic processing. Two unified approaches, optimization and associative retrieval/classification, are demonstrated to be powerful in the majority of the algorithms in these two applications [95].

3. A systematic mapping methodology is adopted to derive systolic/wavefront array architectures for various signal/image and neural nets algorithms [86].

Successful examples include: Kalman filtering for signal tracking [97,98, 99,94], dynamic time warping for speech recognition, stochastic relaxation for image restoration. rank order filtering for image enhancement, Hough transform for shape analysis, 2-D correlation for pattern matching, and 2-D rank decomposition for mask processing [82,100].

4. To facilitate ANN designs using systolic arrays, parallel (synchronous) updating of both discrete and continuous states Hopfield models is proposed, and the convergence issues are discussed [95].

5. Ring systolic array architectures are proposed for both the retrieving and learning phases of various neural net models, including single-layer feedback and multilayer feed-forward nets [99,93,96]. Efficient VLSI implementations of these architectures based on CORDIC techniques are also discussed.

6. A unifying viewpoint between a multilayer Perceptron and a hidden Markov model is proposed [57]. Hidden Markov models described by trellis structures can be regarded as special configurations of multilayer Perceptrons. Iterative gradient-descent-like approaches are successfully applied to the trellis structure to derive the Baum-Welch reestimation formulation.

7. Ring systolic array architectures are proposed for both the scoring and learning phases of hidden Markov models, including both the general and left-to-right models [58,59].

Chapter 2

Algorithms for Artificial Neural Nets

Popular single-layer feedback neural nets (see Figure 1.2(a)) are the Hopfield neural nets [52,53,54,156], the Rumelhart memory/learning modules [117], and the Boltzmann machine [1,51]. On the other hand, feed-forward multilayer Perceptrons (see Figure 1.2(b)) have demonstrated very powerful retrieving/learning capabilities [138,121,139,109].

In addition, there are other varieties of neural nets worthy of further investigation, such as the adaptive resonance theory model [25], the absolute stability model [29], the self-organizing feature map [76], the Neocognitron [39], and structured pattern recognition [123,128].

2.1 Hopfield Neural Nets

Hopfield neural nets have found many applications, including pattern recognition [36], vowel classification [12], combinatorial optimization [54,154], linear programming problem [156], computational vision [73], and parameter estimation [135].

2.1.1 Retrieving Phase

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 [53], Hopfield adopted the following continuous *sigmoid activation* function $f(x)$,

$$f(x) = \frac{1}{1 + e^{-x/u_0}} \tag{2.1}$$

where u_0 controls the nonlinearity of the function. The following system dynamics, with time indices $\{k\}$, were proposed for the retrieving phase [53,154]:

$$\begin{aligned} u_i(k+1) &= \sum_j w_{ij} a_j(k) + \theta_i & (2.2) \\ a_i(k+1) &= \frac{1}{1 + e^{-u_i'(k+1)/u_0}} & (2.3) \end{aligned}$$

where $u_i'(k+1) = \kappa_1 u_i(k) + \kappa_2 u_i(k+1)$ introduced the delay effect in the net input, and κ_1 and κ_2 are proper constants as proposed in [154]. Note that when the coefficient u_0 tends to zero, then the activation function of the Hopfield neural net becomes a *step function* as given in Eq. 2.4:

$$a_i(k+1) = \begin{cases} 1 & \text{if } u_i(k+1) > 0 \\ 0 & \text{if } u_i(k+1) < 0 \\ a_i(k) & \text{if } u_i(k+1) = 0 \end{cases} \tag{2.4}$$

In the special case with no time delay, i.e., $\kappa_1 = 0$ and $\kappa_2 = 1$, the model further reduces to a discrete-state neural net where only two neuron states, 0 and 1, are allowed.

2.1.2 Sequential (Asynchronous) Updating in Hopfield Neural Nets

A main feature of the associative memory is that there exist a number of locally stable points (called attractors) in the state space and all other points in state space fl[illegible] in to the attractors. A common way to guarantee convergence is to find a Liapunov function, i.e a function E on the state space which is nonincreasing along the trajectories. Mathematically, a discrete-time Liapunov function $E(\mathbf{a}(k))$ is defined for a trajectory $\mathbf{a}(k+1) = F(\mathbf{a}(k))$ so that:

$$E(\mathbf{a}(k+1)) \leq E(\mathbf{a}(k))$$

Hopfield claimed that if the sequential (asynchronous) updating is adopted (i.e., at each iteration only one neuron is updated), then the state will change only if the change results in a decrease of the energy level. Therefore, the final convergent state will represent a *local minimum* of the energy function given in in Eq. 1.6. The proof is briefly presented below: To demonstrate the convergence of the iterations, an energy function (Liapunov function in Eq. 1.6) is introduced [53,54]. Let assume that after the k-th updating, we have an energy function $E(k)$:

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

One more updating will lead us to a new energy level $E(k+1)$. Without loss of generality, we first assume that all the neurons are updated in one iteration, i.e., parallel (synchronous) updating, then the energy difference between these two iterations is defined as

$$\begin{aligned}
\Delta_k E &= E(k+1) - E(k) \\
&= -\frac{1}{2}\sum_i \sum_j w_{ij} a_i(k+1) a_j(k+1) - \sum_i \theta_i a_i(k+1) \\
&\quad +\frac{1}{2}\sum_i \sum_j w_{ij} a_i(k) a_j(k) + \sum_i \theta_i a_i(k) \\
&= -\sum_i (a_i(k+1) - a_i(k)) \left[\sum_j w_{ij} a_j(k) + \theta_i\right] \\
&\quad -\frac{1}{2}\sum_i (a_i(k+1) - a_i(k)) \left[\sum_j w_{ij} (a_j(k+1) - a_j(k))\right] \\
&= -\mathbf{u}(k+1)^T \Delta \mathbf{a}(k+1) - \frac{1}{2} \Delta \mathbf{a}^T(k+1) \mathbf{W} \Delta \mathbf{a}(k+1) \\
&= \Delta_k E_1 + \Delta_k E_2 \qquad (2.5)
\end{aligned}$$

where symmetric weights $\{w_{ij} = w_{ji}\}$ are assumed.

In case of sequential (asynchronous) updating, where only one neuron, say i-th neuron, is updated per iteration. As the a_i values evolve, E will also evolve, the change ΔE in E due to changing the state of node i by Δa_i is:

$$\Delta_k E = -u_i(k+1)\ \Delta a_i(k+1) - \frac{1}{2}\ w_{ii}\ \Delta a_i^2(k+1) = \Delta_k E_1 + \Delta_k E_2 \quad (2.6)$$

As long as $w_{ii} \geq 0$, we see that $\Delta_k E_2 \leq 0$. If monotonic nondecreasing activation functions are assumed (e.g., step and sigmoid), then the signs of $\{\Delta a_i(k+1)\}$ and $\{u_i(k+1)\}$ are the same. Thus any change in E under the algorithm is negative. Since E is bounded, the iterations of the algorithm must lead to a stable state that does not need to make any further change.

2.1.3 Weight Determination in Hopfield Neural Nets

To use discrete state Hopfield neural nets for retrieval/classification applications, the unsupervised Hebbian learning rule was adopted by Hopfield to obtain the values of the synaptic weights [52]. In order to store the M binary reference patterns, represented by $\{\mathbf{v}^m = [v_1^m, v_2^m, \ldots, v_N^m], m = 1, 2, \ldots, M\}$, into the associative memory:

$$w_{ij} = \begin{cases} \sum_{m=1}^{M}(2v_i^m - 1)(2v_j^m - 1) & i \neq j \\ 0 & i = j \end{cases} \quad (2.7)$$

Note that $w_{ij} = w_{ji}$, and $w_{ii} = 0$.

To use the continuous-state Hopfield neural nets for optimization applications where the attractors are unknown, the most important step is to formulate the problem into a minimization problem with quadratic cost function, then compare the cost function with the Liapunov energy function given in Eq. 1.6. The synaptic weights for the network can thus be determined. Then the retrieving phase can be performed by following the system dynamics given in Eq. 2.2, 2.3.

2.1.4 Parallel (Synchronous) Updating in the Hopfield Neural Nets

In the original discrete state Hopfield neural net for association tasks, the diagonal weights w_{ii} are all set to zero to guarantee the convergence in the sequential (asynchronous) updating. In order to allow parallel (synchronous) processing of the model, one possible approach is to redefine

$$\begin{aligned} w_{ij} &= \sum_{m=1}^{M} (2v_i^m - 1)(2v_j^m - 1) \\ \mathbf{W} &= \sum_{m=1}^{M} [2\mathbf{v}^{(m)} - \mathbf{1}]\, [2\mathbf{v}^{(m)} - \mathbf{1}]^T \end{aligned} \tag{2.8}$$

From the proof of asynchronous updating, it can be easily shown that $\Delta_k E_1 \leq 0$. Since the $\mathbf{W}$ matrix is formed by an outer product without diagonal nullification (see Eq. 2.8), it is a nonnegative definite matrix, therefore $\Delta_k E_2 \leq 0$.

In the continuous state, the convergence of parallel updating can be assured only when the step coefficient κ_2 is small enough [127]. This will lead us to small values of $\Delta\mathbf{a}(k+1)$. Hence $\Delta_k E_1$ dominates the level of the energy change and the contribution of the energy term $\Delta_k E_2$ in Eq. 2.5 can be neglected.

2.2 Rumelhart Memory/Learning Module

Rumelhart's memory/learning module is another example of single-layer feedback neural net with parallel updating mechanism [117].

2.2.1 Retrieving Phase

The neurons used in the memory/learning modules have continuous activation values ranges from -1 to $+1$. The weights on the connections can be any real values: positive, negative, or zero. At the k-th iteration, the states $a_i(k)$ are updated synchronously (in parallel):

$$u_i(k+1) = \sum_j w_{ij} a_j(k) + \theta_i \tag{2.9}$$

$$a_i(k+1) = \begin{cases} \kappa_1\ u_i(k+1)\ [1 - a_i(k)] - \kappa_2\ a_i(\kappa) & \text{if } u_i(k+1) > 0 \\ \kappa_1\ u_i(k+1)\ [1 + a_i(k)] - \kappa_2\ a_i(k) & \text{if } u_i(k+1) \leq 0 \end{cases} \tag{2.10}$$

where the parameters κ_1 and κ_2 determine the rates of excitation and decay, respectively. Instead of the sigmoid activation, this net uses the ***nonlinear squashing activation function*** given in Eq. 2.10.

2.2.2 Learning Phase

This neural net extracts regularities from an ensemble of inputs without the aid of a sophisticated generalization or rule-formulating mechanism to oversee the performance of the processing system [117]. To adjust the weights in the presentation of m-th training pattern, the net adopts a simple variant of the delta learning rule shown below [117].

$$\begin{aligned} \delta_i^{(m)} &= \theta_i^{(m)} - \sum_{j \neq i} w_{ij}^{(m)} a_j^* \\ w_{ij}^{(m+1)} &= w_{ij}^{(m)} + \Delta w_{ij}^{(m)} \\ &= w_{ij}^{(m)} + \eta \, \delta_i^{(m)} \, a_j^* \end{aligned} \tag{2.11}$$

where a_j^* denotes the converged activation value of j-th neuron in the presentation of m-th training pattern.

2.3 Boltzmann Machine

If we replace the sigmoid coefficient u_0 in the Hopfield neural nets (see Eq. 2.3) by a temperature control parameter T, then Eq. 2.3 becomes

$$\begin{aligned} a_i(k+1) &= f[u_i(k+1)] \\ &= \frac{1}{1 + e^{-u_i(k+1)/T}} \end{aligned} \tag{2.12}$$

where no time delay is introduced.

2.3.1 Retrieving Phase

Let us confine the neuron response to be binary valued (0 or 1), and let the deterministic activation operation in the Hopfield neural net be replaced by a stochastic process defined below.

$$u_i(k+1) = \sum_j w_{ij} a_j(k) + \theta_i \tag{2.13}$$

$$Pr(a_i(k+1) = 1) = \frac{1}{1 + e^{-u_i(k+1)/T}} \tag{2.14}$$

More precisely, $a_i(k+1)$ is set to "1" with probability given in Eq. 2.14, otherwise $a_i(k+1) = 0$. It was shown in [51] that the system dynamics given in Eqs. 2.13 and 2.14 ensure 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 net is called the *Boltzmann machine* [1,51]. If T follows an ***annealing schedule***, i.e. T gradually decreases during the iterations of the neural net, the sigmoid activation operation in Eq. 2.3 becomes an ***annealing*** operation. With a proper annealing schedule, the states of the Boltzmann machine will gradually move toward the global optimal solution.

2.3.2 Weight Determination

Using Boltzmann machine for solving global optimization problems, the weight determination procedure is basically the same as that for the Hopfield net, i.e., starting with formulating the cost function. The synaptic weights can be determined by comparing this cost function with Eq. 1.6. The retrieving phase can be performed by the system dynamics of Eqs. 2.13, 2.14.

2.3.3 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 [140,1,4]: In the first step (denoted *phase*$^+$), some outputs of the neural units (visible units) in the neural net are clamped to predetermined desired values, and then the rest of the neural units (hidden units) iterate through a proper annealing schedule based on Eqs. 2.13 and 2.14 until convergence. In the second step (denoted *phase*$^-$), none of the neural units are clamped and the neural net 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:

$$\Delta w_{ij} = \eta \, (p_{ij}^+ - p_{ij}^-) \tag{2.15}$$

where p_{ij}^+ (respectively, p_{ij}^-) represents the correlation between the two neurons i and j in the *phase*$^+$ (respectively, in the *phase*$^-$). These correlations can be determined by the average probability that both neurons are on (i.e., $a_i^{(m)} = a_j^{(m)} = 1$) over many training patterns ($m = 1, 2, \ldots, M$). Instead of modifying the weights immediately after the exposure of each training pattern, as in the conventional Hebbian learning rule (see Eq. 5.6), the weight adjusting (i.e., Eq. 2.15) 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.

2.4 Multilayer Perceptron

Multilayer Perceptrons are feed-forward neural nets, which have one or more layers of *hidden neuron units* between the input and output neuron layers [123, 121]. For example, a 3-layer net is shown in Figure 1.2(b), where the three layers consist of the two hidden layers and the output layer. Multilayer Perceptrons have been demonstrated to be useful in various applications [103,24,143,109,41].

2.4.1 Retrieving Phase

The mathematical formulation of the multilayer Perceptron basically follows the single layer feedback case, except that the iteration time is now replaced by the layer number. That is the time indices $\{k\}$ previously representing the iteration number in a single layer feedback net are now replaced by the spatial indices $\{l\}$ representing the layer number. The system dynamics in the retrieving phase of an L-layer neural net can be described by the following equations:

$$\begin{aligned} u_i(l+1) &= \sum_{j=1}^{N_l} w_{ij}(l+1)a_j(l) + \theta_i(l+1) && (2.16)\\ a_i(l+1) &= f(u_i(l+1)) \quad 1 \le i \le N_{l+1}, \quad 0 \le l \le L-1 && (2.17) \end{aligned}$$

2.4.2 Learning Phase

The learning phase of a multilayer Perceptron adopts the back propagation learning rule, an iterative gradient descent algorithm designed to minimize the mean squared error between the the desired target values and the actual output values

[140] (for a more detailed derivation, please refer to Section 6.1.3).

The learning rule involves two steps (here we assume that the m-th training pattern is used): (1) In the *forward step*, the input training data are propagated forward across the multi-layer net and the activation values of all the layers $\{a_i^{(m)}(l),\ l = 1, 2, \ldots,\ L\}$ are computed (same as in retrieving phase) according to Eqs. 2.16, 2.17; (2) In the *backward step*, the actual outputs $\{a_i^{(m)}(L)\}$ are compared with the target values $\{t_i^{(m)}\}$ to derive the error signals $\{\delta_i^{(m)}(L)\}$. These error signals are propagated backward layer by layer to be used recursively for the updating of the synaptic weights in all the lower layers. More elaborately, the synaptic weights between the (l-th and $(l-1)$-th) layers can be updated recursively (in the order of $l = L,\ L-1,\ \ldots,\ 1$):

$$\begin{aligned} w_{ij}^{(m+1)}(l) &= w_{ij}^{(m)}(l) + \Delta w_{ij}^{(m)}(l) \\ &= w_{ij}^{(m)}(l) + \eta\ \delta_i^{(m)}(l)\ a_j^{(m)}(l-1) \end{aligned} \tag{2.18}$$

where the error signals $\delta_i^{(m)}(l)$ are recursively computed as follows (again in the order of $l = L, L-1, \ldots, 1$):

$$\text{if } l = L, \quad \delta_i^{(m)}(L) = f'(u_i^{(m)}(L))\ (t_i^{(m)} - a_i^{(m)}(L)) \tag{2.19}$$

$$\text{if } l < L, \quad \delta_i^{(m)}(l) = f'(u_i^{(m)}(l)) \sum_j \delta_j^{(m)}(l+1)\ w_{ji}^{(m)}(l+1) \tag{2.20}$$

where $f'(u_i^{(m)}(l))$ denotes the derivative function of $f(u_i^{(m)}(l))$. If the nonlinear activation function f of Eq. 2.17 is the sigmoid function with unity gain, i.e.,

$$\begin{aligned} a_i(l) &= f(u_i(l)) \\ &= \frac{1}{1+\epsilon^{-u_i(l)}} \\ &= \frac{1}{2}(1 + tanh\ \frac{u_i(l)}{2}) \end{aligned}$$

then the error signals $\{\delta_i^{(m)}(l)\}$ can be simplified as:

$$\delta_i^{(m)}(L) = a_i^{(m)}(L)\,(1-a_i^{(m)}(L))\,(t_i^{(m)} - a_i^{(m)}(L))$$

$$\delta_i^{(m)}(l) = a_i^{(m)}(l)\,(1-a_i^{(m)}(l))\,\sum_j \delta_j^{(m)}(l+1)w_{ji}^{(m)}(l+1)$$

It is suggested in [103] that the ability of multilayer Perceptrons to infer the *inherent rule* is a kin ' of real valued function interpolation. The neural net approximates an arbitrary function by using weighted sum of various nonlinear sigmoid functions, i.e.. weighted sum of some basis functions "bumps," which is analogous to the method of splines for approximating arbitrary functions [15]. Just like splines, the smooth hyperbolic-tangent *tanh* functions are constructed in such a way to maximize smoothness. By taking advantage of the capabilities of this real valued function interpolation, two classes of promising applications of multilayer Perceptrons can be observed, associative retrieval and associative classification.

Chapter 3

Algebraic Projection Analysis of Back-Propagation Learning

Two critical and important issues in back propagation learning are discussed in this chapter: the selection of the optimal number of neurons in each hidden layer and the optimal learning rate in the gradient descent search [50]. The number of hidden units must be sufficient to provide the discriminating capability required by the given application [123], but an excessively large number of synaptic weights may lead to costly and unreliable training. Therefore, it is very desirable to have an *a priori* estimate of the optimal number of hidden neurons. This paper proposes an algebraic projection analysis—an adapting scheme based on the "minimum perturbation principle"—to provide an analytical insight to the problem. More significantly, the same analysis also allows the designer to determine the optimal learning rate in which learning occurs as quickly as possible without incurring instability in the iterative computation.

3.1 Retrieving/Learning Phases of Two-Layer Perceptrons

3.1.1 Retrieving Phase

Without loss of generality, a two-layer Perceptron is used for the following discussion (see Figure 3.1). The system dynamics in the retrieving phase of a two-layer neural net can be described by the following equations:

$$\begin{aligned}
\underline{u}_i &= \sum_{j=1}^{N_0} \underline{w}_{ij} x_j + \underline{\theta}_i = \sum_{j=1}^{N_0+1} \underline{w}_{ij} x_j \\
a_i &= f(\underline{u}_i) \\
\overline{u}_i &= \sum_{j=1}^{P} \overline{w}_{ij} a_j + \overline{\theta}_i = \sum_{j=1}^{P+1} \overline{w}_{ij} a_j \\
y_i &= f(\quad) \quad i = 1,\ 2,\ \ldots,\ N
\end{aligned} \tag{3.1}$$

where $\{\underline{w}_{ij},\ \underline{u}_i,\ \underline{\theta}_i\}$ denote the parameters used in the lower layer, while $\{\overline{w}_{ij},\ \overline{u}_i,\ \overline{\theta}_i\}$ denote the parameters used in the upper layer.

The notation is explained as follows: f denotes the nonlinear (differentiable) activation function: $\{x_i\}$ denote the input patterns; $\{a_i\}$ denote the activation values of the i-th neuron in the hidden layer; $\{y_i\}$ denote the activation values in the output layer; $\{\overline{\theta}_i\}$ ($\{\underline{\theta}_i\}$) denote the internal offsets of neurons in the output (hidden) layer; and $\{\overline{w}_{ij}\}$ ($\{\underline{w}_{ij}\}$) denote the synaptic weights between the output (hidden) layer and the hidden (input) layer of neurons. For simplicity, the internal offset $\overline{\theta}_i$ ($\underline{\theta}_i$) is treated as a synaptic weight $\overline{w}_{i,P+1}$ ($\underline{w}_{i,N_0+1}$), which has a clamped input $a_{P+1} = 1$ ($x_{N_0+1} = 1$). Suppose that there are M distinct training patterns,

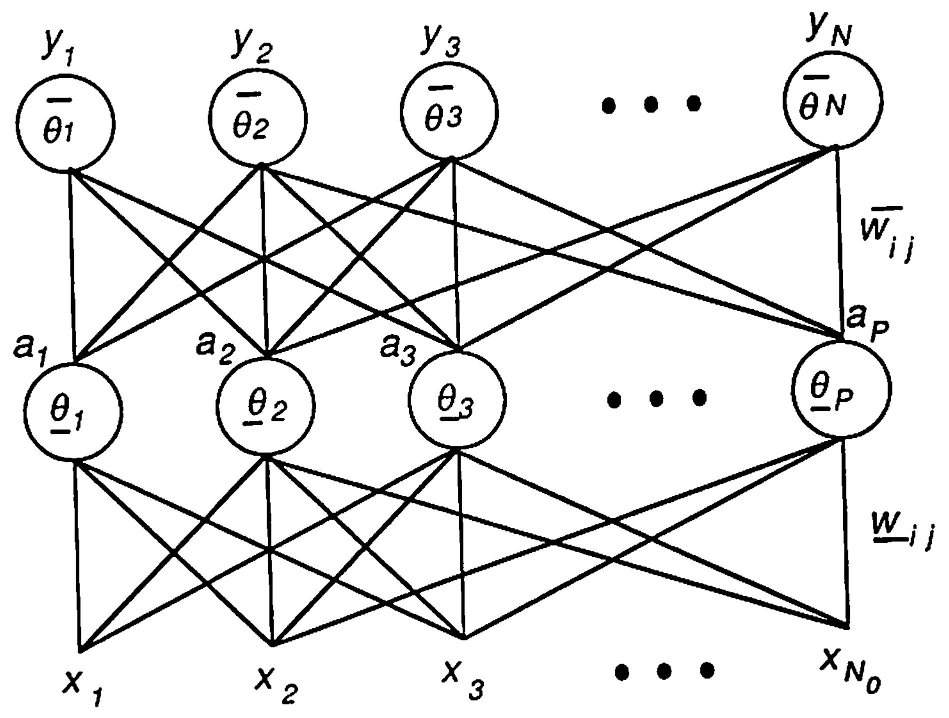


Figure 3.1: A two-layer Perceptron with one hidden layer.

each consist of a pairs of input and target patterns $\{\mathbf{x}^{(m)}, \mathbf{t}^{(m)}\}$. Moreover, the m-th input training pattern $\mathbf{x}^{(m)} = (x_1^{(m)}, x_2^{(m)}, \ldots, x_{N_0+1}^{(m)})^T$. yields the activation values of hidden layer $\mathbf{a}^{(m)} = (a_1^{(m)}, a_2^{(m)}, \ldots, a_{P+1}^{(m)})^T$ and the activation values of output layer $\mathbf{y}^{(m)} = (y_1^{(m)}, y_2^{(m)}, \ldots, y_N^{(m)})^T$.

3.1.2 Learning Phase

The learning phase of a multilayer Perceptron adopts the BP learning rule, which shares the common formulation of most learning rules (assume that the m-th training pattern out of total M patterns is cyclically presented):

$$w_{ij}^{(m+1)} = w_{ij}^{(m)} + \Delta w_{ij}^{(m)} \quad m = 1, 2, \cdots, M, 1, 2, \cdots \tag{3.2}$$

Here $\{w_{ij}^{(m)}\}$ can be the synaptic weight between any two adjacent layers, $\{w_{ij}^{(0)}\}$ specifies the initial weight, and

$$\Delta w_{ij}^{(m)} \propto -\frac{\partial \zeta}{\partial w_{ij}}$$

The error function ζ is defined as

$$\zeta = \sum_{m=1}^{M} \sum_{i=1}^{N} (t_i^{(m)} - y_i^{(m)})^2 \tag{3.3}$$

More specifically,

$$\begin{aligned} \Delta \overline{w}_{ij}^{(m)} &= \eta \, \overline{\delta}_i^{(m)} \, a_j^{(m)} \\ &= \eta \, f'(\overline{u}_i^{(m)}) \, (t_i^{(m)} - y_i^{(m)}) \, a_j^{(m)} \end{aligned} \tag{3.4}$$

$$\begin{aligned}\Delta \underline{w}_{ij}^{(m)} &= \eta\, \underline{\delta}_i^{(m)}\, x_j^{(m)} \\ &= \eta\, f'(\underline{u}_i^{(m)})\, (\sum_{q=1}^{N} \overline{\delta}_q^{(m)}\, \overline{w}_{qi}^{(m)})\, x_j^{(m)} \qquad (3.5)\end{aligned}$$

Each "training sweep" consists of the presentation of all the M training patterns. Usually a large number of training sweeps (most of time in cyclic order) are required for accurate learning. Note that the *updating step* ($\{\Delta \overline{w}_{ij}^{(m)}\}$ or $\{\Delta \underline{w}_{ij}^{(m)}\}$) in the BP learning is linearly proportional to f'. In the following discussion, the nonlinear activation function f adopts the widely used sigmoid function [139],

$$f(u) = \frac{1}{1+e^{-u}} = \frac{1}{2}\,(1 + \tanh u/2)$$

The derivative term then simply reduces to

$$f'(u) = f(u)\,(1 - f(u))$$

3.2 Algebraic Projection Analysis – Kaczmarz Method

Let us now establish the relationship between traditional BP learning and algebraic projection (AP) analysis. Since the nonlinear sigmoid activation function is monotonically increasing, we can define a new set of the transformed target values $b_i^{(m)}$:

$$b_i^{(m)} = f^{-1}(t_i^{(m)}), \quad i = 1, 2, \cdots, N, \quad m = 1, 2, \cdots, M.$$

From basic calculus, if $b_i^{(m)}$ is close enough to $\overline{u}_i^{(m)}$ (which is true in the later stage of the learning process that fine tuning of the weights are performed), then

$$t_i^{(m)} - y_i^{(m)} = f(b_i^{(m)}) - f(\overline{u}_i^{(m)}) \approx f'(\overline{u}_i^{(m)})\,(b_i^{(m)} - \overline{u}_i^{(m)}) \tag{3.6}$$

Using this approximation, the error function ζ in Eq. 3.3 can be rewritten as

$$\begin{aligned}\zeta &= \sum_{m=1}^{M}\sum_{i=1}^{N}(t_i^{(m)} - y_i^{(m)})^2 \\ &\approx \sum_{m=1}^{M}\sum_{i=1}^{N} f'(\overline{u}_i^{(m)})^2\,(b_i^{(m)} - \overline{u}_i^{(m)})^2\end{aligned} \tag{3.7}$$

If we further assume $\{f'(\overline{u}_i^{(m)})\}$ to be constants, a new formulation of the mean squared error in the output layer can thus be defined

$$\begin{aligned}\hat{\zeta} &= \sum_{m=1}^{M}\sum_{i=1}^{N}(b_i^{(m)} - \overline{u}_i^{(m)})^2 \\ &= \sum_{m=1}^{M}\sum_{i=1}^{N}(b_i^{(m)} - \sum_{q=1}^{P+1}\overline{w}_{iq}^{(m)}a_q^{(m)})^2\end{aligned} \tag{3.8}$$

As far as the weight determinations of the output layer are concerned, the minimization of the overall mean square error $\hat{\zeta}$ is equivalent to independently minimizing the mean squared error measure for the individual neuron at the output layer,

$$\hat{\zeta}_i = \sum_{m=1}^{M}(b_i^{(m)} - \sum_{q=1}^{P+1}\overline{w}_{iq}^{(m)}a_q^{(m)})^2 \quad i = 1,\ 2, \ldots,\ N \tag{3.9}$$

Now let us form an $M \times (P+1)$ matrix $\mathbf{A}$, whose rows are the vectors $\{\mathbf{a}^{(m)}\}$, i.e., $\mathbf{A} = [\mathbf{a}^{(1)T},\ \mathbf{a}^{(2)T},\ \ldots,\ \mathbf{a}^{(M)T}\ ]^T$. Similarly, an M-dim vector $\mathbf{b}_i =$

$[b_i^{(1)}, b_i^{(2)}, \ldots, b_i^{(M)}]^T$. To minimize the error function ζ_i defined in Eq. 3.9, one can solve the following linear system, which corresponds to a least-square error system

$$\mathbf{A}\mathbf{w}_i = \mathbf{b}_i \tag{3.10}$$

where $\mathbf{w}$ is the $(P+1)$-dim vector with its elements being $\{\overline{w}_{ij},\ j = 1, 2, \ldots, P+1\}$. Note that here we only deal with the $P+1$ synaptic weights linking the i-th neuron in the output layer. The same considerations can be carried through to all the other weights linking other neurons in the output layer.

For solving the linear system given in Eq. 3.10, an iterative algebraic projection (AP) scheme—called the Kaczmarz method—can be adopted [27,155]. This AP scheme is a *row action* method originally proposed for solving huge and sparse linear systems [27]. Basically, this method starts from an initial vector $\mathbf{w}_i^{(0)}$ and iteratively converges to the solution vector $\mathbf{w}_i^*$. In one iteration step, the vector $\mathbf{w}_i^{(m)}$ is refined into a new vector $\mathbf{w}_i^{(m+1)}$ which may better match the m-th row of the linear system in Eq. 3.10.

$$\begin{aligned}
\overline{w}_{ij}^{(m+1)} &= \overline{w}_{ij}^{(m)} + \Delta\overline{w}_{ij}^{(m)} \\
&= \overline{w}_{ij}^{(m)} + \eta'\ (b_i^{(m)} - \sum_{q=1}^{P+1} \overline{w}_{iq}^{(m)} a_q^{(m)})\ a_j^{(m)} \\
&= \overline{w}_{ij}^{(m)} + \eta'\ (b_i^{(m)} - \overline{u}_i^{(m)}) a_j^{(m)}
\end{aligned} \tag{3.11}$$

where the learning rate η' of the AP scheme is defined as:

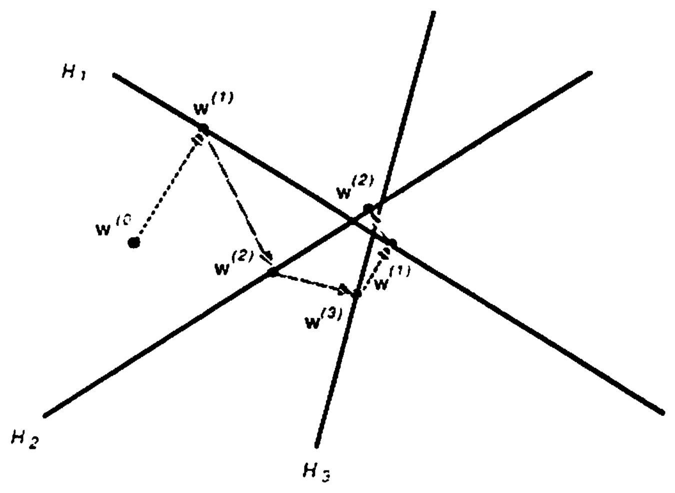


Figure 3.2: The geometric interpretation of the AP scheme for solving a 3×2 linear system. Here the relaxation parameter λ is 1.

$$\eta' = \frac{\lambda}{\sum_{q=1}^{P+1} (a_q^{(m)})^2} \tag{3.12}$$

and λ is the relaxation parameter, $0 < \lambda < 2$. For unity relaxation ($\lambda = 1$), the geometric interpretation of this AP method is provided in Figure 3.2. Given a $(P+1)$-dimensional point vector $\mathbf{w}_i^{(m)}$, then $\mathbf{w}_i^{(m+1)}$ is the orthogonal projection of $\mathbf{w}_i^{(m)}$ onto the P-dimensional hyperplane subspace H_m (see Figure 3.2), where the m-th hyperplanes H_m is defined as:

$$H_m = \{\mathbf{w}_i \in R^{P+1} \mid \sum_{q=1}^{P+1} a_q^{(m)} w_{iq} = b_i^{(m)}\} \tag{3.13}$$

3.2.1 Exponential Convergence of the AP Method

The exponential convergence behavior of the iterated weight vector $\mathbf{w}_i^{(m)}$ during the AP learning is similar to the normalized LMS techniques used in the adaptive signal processing [71,171]. A simple description of the exponential convergence behavior of the AP method is given below [21]. Let us first assume that a consistent (exact) solution $\mathbf{w}_i^*$ can be obtained, i.e., $\mathbf{A}\mathbf{w}_i^* = \mathbf{b}_i$. A *difference variable* $\mathbf{v}^{(m+1)}$ is defined, which specifies the difference between the consistent solution $\mathbf{w}_i^*$ and the actual iterated estimate $\mathbf{w}_i^{(m+1)}$ in the $(m+1)$-th iteration.

$$\begin{aligned}
\mathbf{v}^{(m+1)} &= \mathbf{w}_i^* - \mathbf{w}_i^{(m+1)} \\
&= \mathbf{w}_i^* - \mathbf{w}_i^{(m)} - \lambda\,(b_i^{(m)} - \mathbf{a}^{(m)T}\mathbf{w}_i^{(m)})\,\frac{\mathbf{a}^{(m)}}{|\mathbf{a}^{(m)}|^2} \\
&= \mathbf{v}^{(m)} - \lambda\,(\mathbf{a}^{(m)T}\mathbf{w}_i^* - \mathbf{a}^{(m)T}\,\mathbf{w}_i^{(m)})\,\frac{\mathbf{a}^{(m)}}{|\mathbf{a}^{(m)}|^2} \\
&= \mathbf{v}^{(m)} - \lambda\,\mathbf{a}^{(m)T}\,\mathbf{v}^{(m)}\,\frac{\mathbf{a}^{(m)}}{|\mathbf{a}^{(m)}|^2} \\
&= (\,\mathbf{I} - \lambda\,\frac{\mathbf{a}^{(m)}\mathbf{a}^{(m)T}}{|\mathbf{a}^{(m)}|^2}\,)\mathbf{v}^{(m)} \qquad (3.14)
\end{aligned}$$

where $\mathbf{w}_i^{(m)} = [\,\overline{w}_{i1}^{(m)},\ \overline{w}_{i2}^{(m)},\ \cdots,\ \overline{w}_{i,P+1}^{(m)}\,]$. The matrix $\frac{\mathbf{a}^{(m)}\mathbf{a}^{(m)T}}{|\mathbf{a}^{(m)}|^2}$ has a single eigenvalue (or singular value) of *one* and remaining eigenvalues of *zero*,

$$\sigma\{\frac{\mathbf{a}^{(m)}\mathbf{a}^{(m)T}}{|\mathbf{a}^{(m)}|^2}\} = [\,1,\ 0,\ 0,\ \ldots,\ 0\,]$$

where $\sigma(\cdot)$ represents the eigenvalues operator. It can be seen that to guarantee convergence [170]

$$0 < \lambda < 2$$

In situations where a consistent solution is not available, a minimum norm solution is preferred. The minimum norm solution introduces an estimation residual $\mathbf{r}_i = [r_i^{(1)},\ r_i^{(2)},\ \cdots,\ r_i^{(M)}\ ]^T$ into the system equation, $\mathbf{A}\mathbf{w}_i^* + \mathbf{r}_i = \mathbf{b}_i$. Equation 3.14 can thus be rewritten as:

$$\begin{aligned}
\mathbf{v}^{(m+1)} &= \mathbf{w}_i^* - \mathbf{w}_i^{(m+1)} \\
&= \mathbf{w}_i^* - \mathbf{w}_i^{(m)} + \lambda\ (b_i^{(m)} - \mathbf{a}^{(m)T}\mathbf{w}_i^{(m)})\ \frac{\mathbf{a}^{(m)}}{|\mathbf{a}^{(m)}|^2} \\
&= \mathbf{v}^{(m)} - \lambda\ (\mathbf{a}^{(m)T}\mathbf{w}_i^* - \mathbf{a}^{(m)T}\ \mathbf{w}_i^{(m)})\ \frac{\mathbf{a}^{(m)}}{|\mathbf{a}^{(m)}|^2} + \lambda\ r_i^{(m)}\ \frac{\mathbf{a}^{(m)}}{|\mathbf{a}^{(m)}|^2} \\
&= \mathbf{v}^{(m)} - \lambda\ \mathbf{a}^{(m)T}\ \mathbf{v}^{(m)}\ \frac{\mathbf{a}^{(m)}}{|\mathbf{a}^{(m)}|^2} + \lambda\ r_i^{(m)}\ \frac{\mathbf{a}^{(m)}}{|\mathbf{a}^{(m)}|^2} \\
&= (\ \mathbf{I} - \lambda\ \frac{\mathbf{a}^{(m)}\mathbf{a}^{(m)T}}{|\mathbf{a}^{(m)}|^2}\ )\mathbf{v}^{(m)} + \lambda\ r_i^{(m)}\ \frac{\mathbf{a}^{(m)}}{|\mathbf{a}^{(m)}|^2} \qquad (3.15)
\end{aligned}$$

In order to avoid the increase of the estimation residual, a practical range for λ is

$$0 < \lambda \leq 1$$

It has been shown that for the unity relaxation case ($\lambda = 1$), the weight updating formulation in Eq. 3.11 will converge to the exact solution when Eq. 3.10 is a consistent linear system, and will converge to a minimum norm solution when an inconsistent or singular system is encountered [155,116].

3.3 Selection of Optimal Number of Hidden Neurons in BP Learning

The number of hidden units per layer dictates the space partitioning separability (and thus the discriminating capabilities) of a multilayer Perceptron [123,121]. The computational complexity is also significantly dependent on the number of hidden units. The number of hidden units must be large enough to form a decision region that is as complex as is required by a given problem. However, if it is too large, then the weights become difficult to reliably estimate from the training data. Therefore, the use of an *optimal* number of hidden units per layer is highly desirable for the purpose of *efficient learning*. Note that our definition of optimality is in terms of learning time required to lead to a convergence. There are some other alternatives can be used, e.g., in terms of fault tolerance or generalization capabilities [148].

In order to understand the roles played by the hidden units, it is useful to link BP learning with the AP scheme to take advantage of the numerous AP studies in the numerical analysis literature. Due to the high similarity between the weight updating formulas in the AP scheme and in BP learning, the numerical convergence properties of the AP scheme can be used to explain the dynamic behavior of BP learning, and furthermore, to help estimate the optimal number of hidden units for BP learning.

3.3.1 Convergence of the A Matrix

It should be noted that within a training sweep, the matrix **A** remains constant. However, the matrix **A** changes from one sweep to the next sweep. Hopefully, after a certain number of sweeps, the matrix **A** will converge to a constant. Then the AP analysis can be adopted to analyze the BP learning rule. This assumption may be justified by some observations reported in [130,55]. When the weight values are close to the desired values, the lower layer weights are changing very slowly and can be assumed to be almost fixed (i.e., **A** is a constant matrix), thus the upper layer weight updating dominates the subsequent stage of learning and seems to consume most of the computation time [130]. In fact, some studies have shown that those lower layer weights (or the partitioning hyperplanes) may be fixed from the very beginning. If the hyperplanes are near certain decision boundaries that are required in a specific problem, tl.' method offers impr. ement in convergence speed [55].

3.3.2 Effective Dimension M^* and Hidden Units Size P

The convergence properties of the AP scheme have been studied [155,71]. It is noted that the solution may wander in the weight space in either the overdetermined case (too few hidden units), or in the underdetermined case (too many hidden units) [92]. Let us define an *effective dimension* M^* to be the effective numerical *cluster* (not rank) formed by the converging matrix **A**, (obviously, $M^* \leq M$). In terms of a geometric interpretation, the effective dimension M^* can be regarded as the distinct *bundles* of hyperplanes created from **A** matrix.

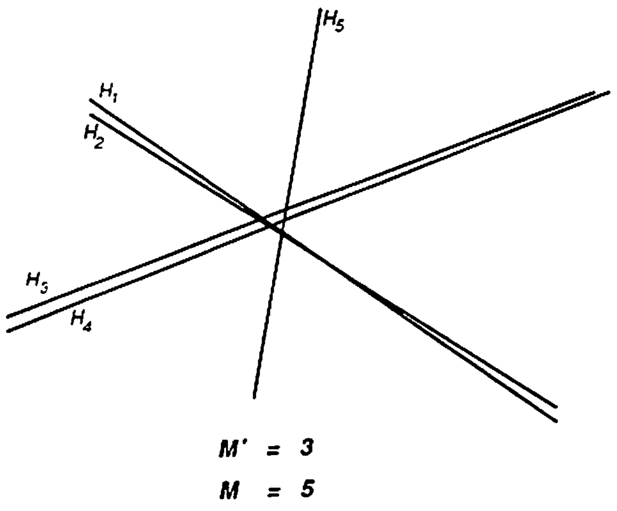


Figure 3.3: There are $M^* = 3$ bundles of distinct hyperplanes (lines) in the presence of $M = 5$ hyperplanes.

For example, in Figure 3.3 there are $M^* = 3$ bundles of distinct hyperplanes (lines) in the presence of $M = 5$ hyperplanes.

Nonsingular Case $M^* = P + 1$ In the case of nonsingular systems ($M^* = P + 1$), where a unique solution exists, the fast convergence of the AP scheme has been shown by Kaczmarz [68]. Similar results have been reported in another theoretical numerical study [155]. This leads to a good choice of the optimal number of hidden neurons. $P = M^* - 1$, especially in the associative retrieval applications which will be discussed momentarily.

Overdetermined Case, $M^* > P + 1$ As shown in Figure 3.2, an overdetermined system ($M^* > P + 1$) is not guaranteed to have a unique solution. In this case, the AP scheme will oscillate in the neighborhood of the region of the intersections of the hyperplanes, which slows down the learning significantly.

Underdetermined case, $M^* < P + 1$ In the underdetermined case ($M^* < P + 1$), where an infinite number of solution are possible, the AP scheme can be forced to converge to a solution $\mathbf{w}_i^{(m)}$, such that $|\mathbf{w}_i^{(m)} - \mathbf{w}_i^{(0)}|$ is minimized. The solution is nonunique, and the system will pick any convenient converging point [155].

3.3.3 Totally Irregular/ Regularly-Embedded Training Patterns

Totally Irregular Training Patterns (Clusters) If the training patterns (clusters) are totally irregular, then the effective dimension M^* should be equal to total number of training patterns (clusters), which can be easily pre-computea from the training patterns. Therefore, the convergence properties of the AP method can be used to predict that the optimal number of hidden neurons P should be approximately equal to $M^* - 1$ for efficient learning. It should be noted again that this conjecture is suitable only for totally "random" or "irregular" training patterns, e.g., most pattern association and pattern classification applications. This result is consistent with the concept of the distributed "grandmother cell" in the associative content addressable memory [18].

Regularly-Embedded Training Patterns For those tasks which have inherent regularities in the training patterns (e.g., orthogonal encoding [139], parity checking [140,159], 3-in-8 detecting [148], etc), the optimal number of hidden units size is still equal to $P = M^* - 1$. However, the determination of the effective dimension M^* is much more involved and very problem dependent.

After some number of training sweeps during learning, regularity is established in the internal representation stored by the hidden units. At this point the BP learning algorithm is supposed to switch from *rote learning* to some kind of *regularity enhancing* or *predicate establishing* process.[1] In this case, the effective dimension M^* is dictated by the number of *regularity features* or the *order of predicate* in the problem, instead of the number of training patterns.

3.3.4 Simulations for Optimal Hidden Units Size

Simulations have been conducted for the verification of the prediction of the optimal hidden units size in various real-valued pattern association and pattern classification tasks using BP learning [101,92].

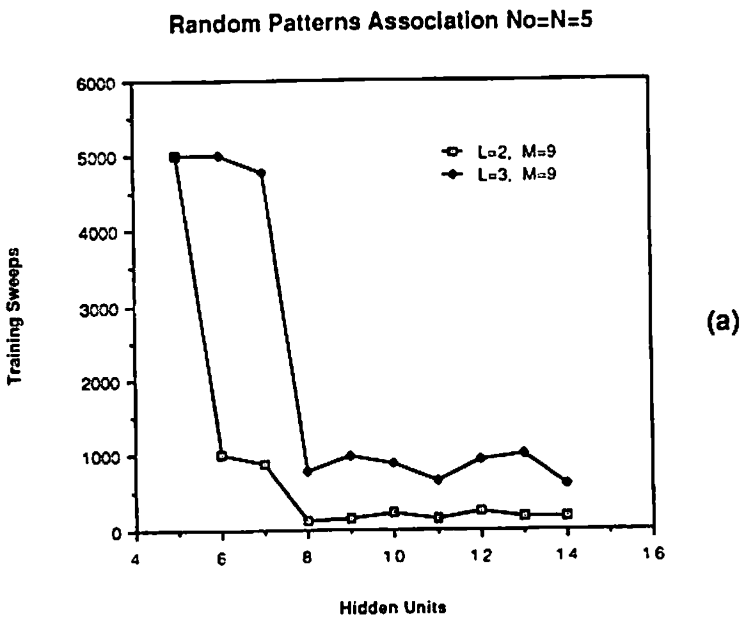
Pattern Association Tasks Simulations were conducted for pattern association tasks with different numbers of layers (2 and 3) and different number of hidden units per layer. Figure 3.4(a) shows the convergence time (i.e., training sweeps) vs. the number of hidden units per layer. In our simulations, the learning convergence criteria is set so that the average error (among all the output neurons and all the training patterns) is below 0.05 per neuron output. There are $M^* = M = 9$ pairs of randomly generated 5-dimensional input and 5-dimensional output distinct training patterns (each dimension contains a random number

[1]The definition of the predicate follows that of in [121].

picked from a uniform distribution between 0 and 1). A 3-layer ($L = 3$) perceptron is used in which the number of the hidden units per layer, which is set to be equal in both hidden layers varies from 5 to 14. We observed that the net with more hidden units per layer leads to a smaller number of training sweeps. The neural nets stopped learning after they converged to the pre-specified mean-squared error accuracy, or after 10^4 training sweeps without reaching the pre-specified mean-squared error accuracy. It is important to note that there is consistently an abrupt reduction of training sweeps around $M^* - 1$ (i.e., 8) hidden units. This matches with the theoretical optimal hidden units predicted by AP analysis. (Note that this simulation also suggests that the AP results appear to be valid for ANNs with more than one hidden layer).

In another simulation, randomly generated 5-dimensional input and output pairs are used as training patterns for a two-layer Perceptron. Two different numbers of pairs of training patterns, i.e., $M^* = M = 9, 19$ are used in this simulation. The simulation results comparing convergence time with the number of hidden units per layer is shown in Figure 3.4(b). Note that the number of training sweeps in these simulations dives consistently when the number of hidden units is near or equal to $M^* - 1$. This further supports the theoretical results of the AP analysis.

Pattern association tasks with clustered input training patterns were also simulated for two layer Perceptrons. In these simulations, 20 clusters of training patterns were used (i.e., M^*=20). Each cluster consisted of 5 similar 10-dimensional



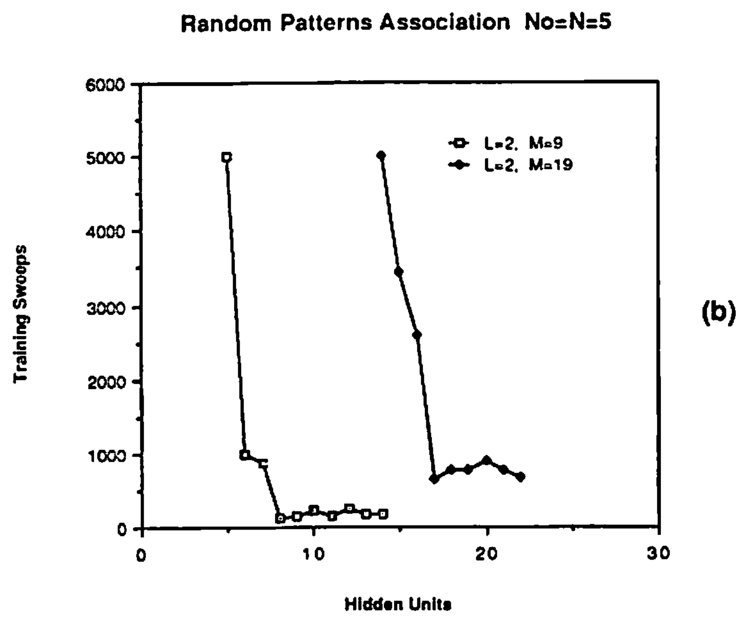


Figure 3.4: (a) The convergence time vs. the number of hidden units with 1 or 2 hidden layers. (b) The convergence time vs. the number of hidden units with various numbers of training patterns.

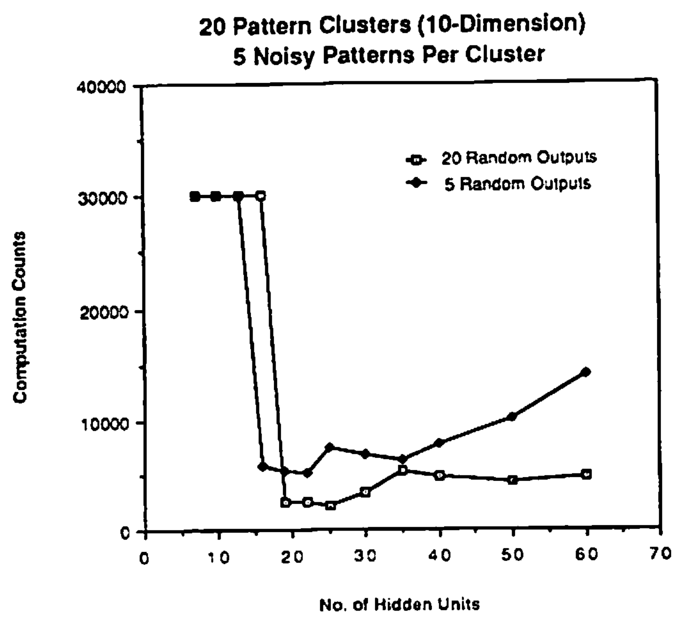


Figure 3.5: Simulation for the optimal selection of hidden unit size in the pattern association tasks. The relative total computation counts vs. the number of hidden units with clustered random training patterns.

input patterns (i.e., M=100) with each dimension containing a random number picked from a uniform distribution between 0 and 1 and corrupted with noise uniformly distributed between -0.1 and 0.1. For all 5 input patterns in the same cluster, there is only one associated output pattern (which is either a 20-dimensional or a 5-dimensional vector with elements being random numbers uniformly distributed between 0 and 1).

For ease of later comparisons with pattern classification tasks, the simulation result is shown in terms of the relative total computation counts (instead of using training sweeps) vs. the number of hidden units (see Figure 3.5). Note that all the lowest computation counts in these simulations fall consistently on the point where the number of hidden units is near or equal to $M^* - 1 = 19$.

Pattern Classification Tasks Pattern classification tasks with clustered input training patterns were also simulated in a two-layer Perceptron. The same set of clustered input training patterns used in the pattern association tasks are used for the pattern classification tasks. In these simulations, each cluster can be specified by one neuron in the output layer (i.e., a 20-dimensional binary output pattern with a single one value), or it can be specified by a binary coded output (i.e., a 5-dimensional binary output pattern).

The simulation results comparing the relative total computation counts vs. the number of hidden units are shown in Figure 3.6. Compared to the random output pattern association tasks as shown in Figure 3.5, the pattern classification tasks are much easier to train. This phenomenon is due to the nonlinearity of the sigmoid function, which maps a very large range of net input values to the two extreme output activation values (i.e., 0 and 1), this results in faster convergence under the same error criterion.

It is interesting to note that due to the binary values (0 or 1) used in the neurons of the output layer, ambiguities are created in defining the corresponding transformed target values $\{b_i^{(m)}\}$. This implies the projection hyperplanes $\{H_i\}$ in Eq. 3.13 have more freedom for horizontal movement, so that the convergence behavior of the algebraic projection can be relaxed. As shown in Figure 3.6, instead of a single optimal number of hidden units size, a larger span (range) of optimal sizes (i.e., lowest computation counts) can be observed around the point where the hidden unit size is near or equal to $M^* - 1 = 19$.

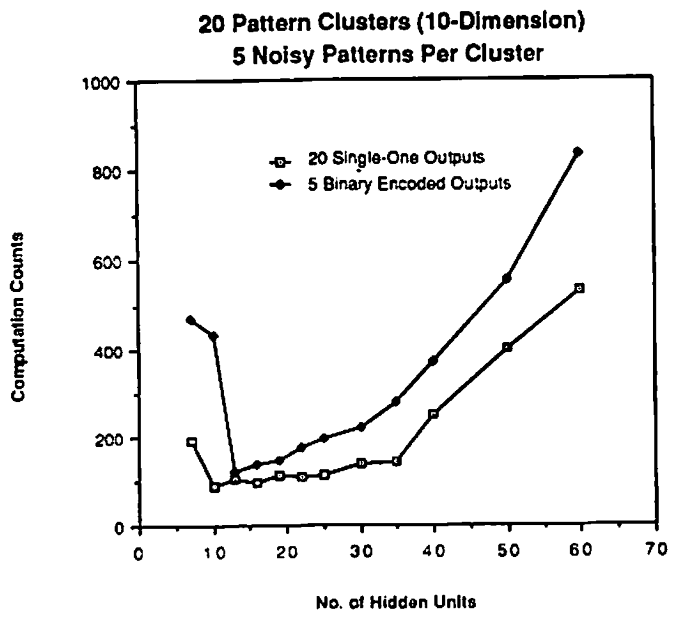


Figure 3.6: Simulation for the optimal selection of hidden unit size in the pattern classification tasks.

3.4 Selection of Optimal Learning Rate in BP Learning

It has been conjectured, based on a fan-in argument, that the learning rate η in BP learning should decrease as $1/N_0$, where N_0 is the number of input dimensions [130]. There is another conjecture, based on empirical studies for the parity checking problem, that the learning rate should decrease somewhat faster than $1/N_0$ [159,50]. Here we propose a systematic approach using AP analysis, to determining an optimal value of the learning rate parameter η in BP learning.

3.4.1 Comparison of η and η'

Based on Eq. 3.6, which gave the relationship between $\{b_i^{(m)} - \overline{u}_i^{(m)}\}$ and $\{t_i^{(m)} - y_i^{(m)}\}$, we can rewrite Eq. 3.11 as:

$$\begin{aligned} \overline{w}_{ij}^{(m+1)} &= \overline{w}_{ij}^{(m)} + \eta' \, (b_i^{(m)} - \overline{u}_i^{(m)}) a_j^{(m)} \\ &\approx \overline{w}_{ij}^{(m)} + \eta' (f'(\overline{u}_i^{(m)}))^{-1} \, (t_i^{(m)} - y_i^{(k+1)}) \, a_j^{(m)} \end{aligned} \tag{3.16}$$

By comparing the two weight updating formulations for the output layer shown in Eq. 3.4 and Eq. 3.16, we note that:

$$\text{BP Updating Step}: \quad \propto \quad \eta \, f' \tag{3.17}$$

$$\text{AP Updating Step}: \quad \propto \quad \eta' (f')^{-1} \tag{3.18}$$

This results in the following relationship between two learning rate:

$$\eta = (f')^{-2} \, \eta' \tag{3.19}$$

As shown in Eq. 3.12, the exact value of η' in AP learning depends on the values of $\{a_q^{(m)}\}$. However, if $\{a_q^{(m)}\}$ are treated as random variables with equal probability (50%) of being 0 and 1 (due to the high nonlinearity), then an *expectation value* of the optimal learning rate $E[\eta']$ for AP learning can be derived as (assuming that the relaxation parameter λ is set to 1)

$$E[\eta'] = \frac{1}{(P+1) \, \mathrm{E}[(a_q^{(m)})^2]} = \frac{2}{P+1}$$

In order to estimate the *expectation value* of the optimal learning rate $E[\eta]$ for BP learning through AP analysis (see Eq. 3.19), the expectation value of the derivative of the nonlinear sigmoid function $E[f']$ should be computed first. Remember that,

$$f(u) = \frac{1}{1 + e^{-u}} \quad \text{and} \quad f'(u) = f(u)\ (1 - f(u))$$

where $0 \leq f(u) \leq 1$. The expectation value of $f'(u)$ can be calculated, i.e.,

$$\begin{aligned} E[f'(u)] &= \int_{-\infty}^{\infty} f'(u)\ Pr(u)\ du \\ &= \int_0^1 f'(f(u))\ Pr(f(u))df(u) \\ &= \int_0^1 f(u)\ (1 - f(u))\ df(u) \\ &= \int_0^1 a\ (1 - a)\ da \\ &= \frac{1}{6} \end{aligned}$$

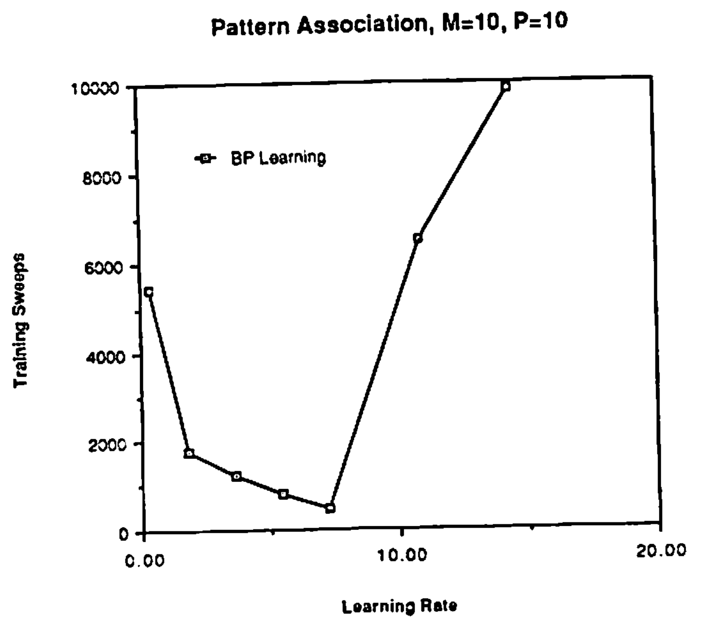
where $Pr(u)$ denotes the probability density function of random variable u, and we assume $f(u)$ is uniformly distributed between 0 and 1, i.e., $Pr(f(u)) = 1$. Now from Eq. 3.19, the expectation value of the optimal learning rate $E[\eta]$ for BP learning can be derived:

$$E[\eta] = E[f']^{-2}\ E[\eta'] = 36\ E[\eta'] = \frac{72}{P+1}$$

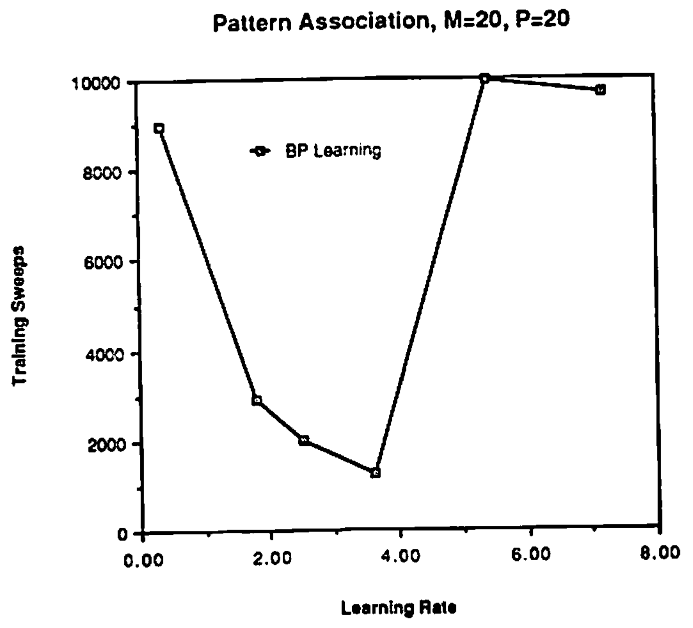
3.4.2 Simulations for the Optimal Learning Rate

Simulations have also been conducted for real-valued random pattern association tasks to test the validity of the conjecture of the optimal learning rate in BP learning. In the following simulations. we used two layers Perceptron in which the size of the hidden units equaled the number of training patterns, i.e., $P = M^* = M$.

Figure 3.7(a) shows the average convergence time vs. various values of the learning rate. There are $M = 10$ pairs of randomly generated 5-dimensional input and 5-dimensional output training patterns (each dimension contains a random number picked from a uniform distribution between 0 and 1) to be trained by BP learning. In our simulations. the algorithm cannot converge (within 10,000 sweeps) to a pre-specified error accuracy if the values of η are outside of the region shown in Figure 3.7(a) and (b). As shown in Figure 3.7(a), the average training time curves dive around the vicinity of $\eta = 7.2$, which is perfectly matched to the optimal learning rate conjecture, i.e., $\eta = \frac{72}{P+1} = \frac{72}{11}$. Simulations using $M = 20$ patterns were also conducted. and also support the learning rate conjecture. Figure 3.7(b) shows the average training time curve dive around the vicinity of $\eta = \frac{72}{P+1} = \frac{72}{21} \approx 3.6$.



(a)



(b)

Figure 3.7: (a) The average convergence time vs. various values of η in BP learning $M = P = 10$. (b) The average convergence time vs. various values of η, $M = P = 20$.

Chapter 4

Applications of Artificial Neural Nets

Research in ANNs involves a very broad spectrum of disciplines, including algorithm analysis, parallelism extraction, array architectures, programming techniques, functional primitives, structural primitives, and the numerical performance of algorithms. In striving for a more cohesive exploration, cross-disciplinary discussions on applications are necessary.

A successful applicational study requires an understanding of the signal formation process, the algorithm class involved, and the specifications of the intended applicational system. The immense computational complexity inherent in mimicking either the human visual perception process in computer vision, or dynamic movement control in robotic processing, exhaust massive amounts of computations. On the other hand, humans perform these tasks easily via biological neural nets. It becomes obvious that two potential applications suitable for ANNs are computer vision and robotic control. Most algorithms in these two applicational domains can be formulated either as optimization (or constrained optimization) problems or pattern association tasks, and thus can be efficiently implemented by ANNs.

4.1 Applications in Computer Vision

Modern computer vision, with the goal of *understanding* images of complex 3-D scenes, was first attempted in the early 1960s. The immense computational complexity inherent in mimicking the human visual process soon became apparent, and devoting massive amount of processing at the early stages of vision was technically and economically impossible, so in the 1970s a cognitive approach to computer vision emerged which conveniently minimized image-level numerical computation and emphasized symbolic manipulation. In the 1980s, the consensus of the computer vision community is that the gap between the early vision and image understanding is best bridged by producing a redundant set of intermediate visual data representations formed before object recognition is attempted. While the human vision system copes very easily with vision by using biological neural nets, computer vision requires a huge amount of computation just to form the intermediate representations [56]. These image-like representations are registered with the input image and contain values of physical parameters of scene points such as the distance from a sensor to a point, the reflectance/brightness of surfaces, the direction of motion of objects, the orientation of surfaces, and so forth. The fact that human vision work so well imply that those neural nets rely on natural constraints or assumptions about the world to derive an unambiguous output [122,15].

It seems that although new algorithms and architectures are making an impact on what can be achieved, there is still a long way to go before artificial systems will compete with the speed and flexibility of the human visual system. The demands of the computation required to perform even very specific visual tasks in real time outstrip current parallel architectures. This fact, together with the requirement for automation involving visual sensors, means that computer vision applications are a major driving force in the development of more powerful neural net algorithms and architectures [124].

Two main goals of computational vision processing are to develop image understanding systems, which automatically construct scene descriptions from image input data, and to understand human vision. Much of the current research has analyzed processes in early vision because the inputs and the goals of the computation can be well characterized at this stage. Several problems have been solved and several specific algorithms have been successfully developed. Examples are stereo matching, motion and optical flow estimation, shape from shading, and surface reconstruction [131]. In the following, stereo matching using constrained optimization techniques implemented by ANNs is demonstrated.

4.1.1 Stereo Matching

Traditionally, sonar sensors have been very popular for depth determination [63]. However, stereo matching techniques are a more reliable but more computationally costly alternative. Stereo matching recovers 3-D depth information from two images taken from two cameras. Usually, the geometry of the imaging system (e.g., the camera positions and their orientations) are known. The main task of stereo vision is to match the feature primitives of the two images. This is called the *correspondence problem*, and consists of detection of feature primitives (e.g., edges [151]) and the determination of valid matches between feature primitives. As discussed next, the latter task may be formulated as a constrained optimization problem [115,151,152,175].

As shown in Figure 4.1, the stereo matching problem involves two cameras with focal length f and 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 the (i, j) pixel in the left image and at the (i', j') pixel in the right image. Without loss of generality, let us assume that the epipolar scanline is along the horizontal (J-axis) direction, so that $i = i'$ and the search for a candidate match for a point in one image can be limited to the same row in the other image. The value $(j - j')$ is called the *disparity* value between two matching points. The depth information z can be generated from the disparity value, i.e. $z = \frac{2fd}{(j-j')}$.

4.1.2 Neural Nets for Constrained Optimization

Consider the following constrained optimization problem [112]:

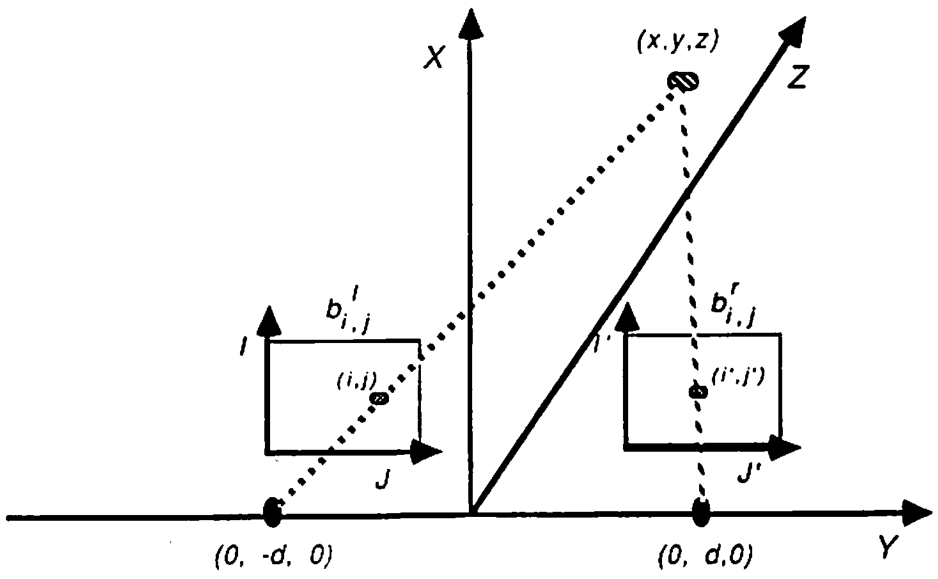


Figure 4.1: A standard stereo matching problem involves two camera with focal length f and their focal points lying at $(0, \pm d, 0)$.

$$\begin{aligned} &\text{minimize} && \phi(\mathrm{a}) \\ &\text{subject to the constraints} && P_i(\mathrm{a}) = 0 \quad i = 1, 2, \cdots, p \end{aligned} \tag{4.1}$$

where ϕ is a continuous function of n-dimensional vector a and $\{P_i(\mathrm{a})\}$ are non-negative 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 [112]:

$$\text{minimize } E' = \phi(\mathrm{a}) + \sum_{i=1}^{p} c_i P_i(\mathrm{a}) \tag{4.2}$$

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

One famous example is the Hopfield neural net for solving the (N cities) traveling salesman problem. In the problem formulation, an array of $N \times N$ neurons are used to solve the problem. The desired solution minimizes the traveling length among N cities, under the constraint that the matrix of neurons should represent a permutation matrix (valid solution).

4.1.3 Neural Net Formulation for Stereo Matching

Suppose that there are two $M \times N$ image intensity arrays, which can be primitives extracted, $\{b^l_{i,j}\}$ and $\{b^r_{i,j}\}$ as taken by the (left and right) cameras. A 3-D binary matching data array $\{a_{i,j,k}, 1 \leq i \leq M, 1 \leq j \leq N, 0 \leq k \leq D\}$ may be defined to represent the status of matching [115,175]. When the $a_{i,j,k}$ is 1, this means that the disparity value is k at the point (i, j). In this discussioı ːhe maximal disparity value is limited to D. The goal of the correspondence problem is to minimize the following cost function φ:

$$\begin{aligned}
\phi &= \phi_1 + \phi_2 \\
&= \sum_{i=1}^{M} \sum_{j=1}^{N} \sum_{k=0}^{D} (b^l_{i,j} - b^r_{i,j\oplus k})^2 \, a_{i,j,k} + \\
&\quad \lambda \sum_{i=1}^{M} \sum_{j=1}^{N} \sum_{k=0}^{D} \sum_{(i',j') \in \Psi} (a_{i,j,k} - a_{i',j',k})^2 \qquad (4.3)
\end{aligned}$$

with the constraint

$$P = \sum_{i=1}^{M} \sum_{j=1}^{N} (\sum_{k=0}^{D} a_{i,j,k} - 1)^2 = 0 \qquad (4.4)$$

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

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

2. ϕ_2 is a measurement of the continuity of the depth value [115,175], where λ is a proper weighting constant and Ψ denotes the the neighborhood around the pixel (i,j), defined by a $q \times q$ window centered at (i,j).

3. The constraint P in Eq. 4.4 is introduced to assure the *uniqueness* property is preserved. That is any point from each image can assume one and only one depth value [115.15].

Following Eqs. 4.2, 4.3, and 4.4, the constrained problem can be reformulated as an unconstrained problem, i.e.,

$$\min_{a_{i,j,k}} E' = \phi + cP \tag{4.5}$$

The Hopfield neural net 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 Sec. 1.3.2, the synaptic weights $w_{i,j,k,l,m,n}$ and the external input $\theta_{i,j,k}$ of the neural net can be derived by equating the energy (penalty) function in Eq. 4.5 to the Hopfield energy function E in Eq. 1.6 [54,175], i.e.,

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

4.2 Applications in Robotic Control

There are three levels of processing in a hierarchical robotic control system: task planning, path planning, and path control [145]. In the task planning level, the robot will receive instructions about task plans and manage information (e.g., target depth, obstacle locations) about the workspace. At the path planning level, a sequence of desired path trajectories (e.g., robot positions, arm orientations) is produced. The path control level generates the necessary motor commands (torques and forces) in the joint coordinates to drive the robot or robot arm to follow the desired trajectory.

The computational requirements of robotic control are very demanding. At the level of task planning, primitive image/vision analysis is required to obtain 3-D information. At the level of path planning, an optimal path in a workspace has to be selected by using optimization techniques. In the path control level, real-time computing of inverse dynamics and kinematics must be provided.

Neural net techniques for processing the various algorithms in the three levels of robotic control have been proposed. The approach is to formulate each algorithm either as an optimization problem or a pattern association problem. These algorithms include stereo vision in task planning [115,151,152,175], autonomous robot path planning [161,64], manipulator position control [163,132], and robotic voluntary movement control [70]. In the following, two examples are provided as demonstrations of the use of ANNs in robotic control.

4.2.1 Path Planning Using Neural Nets

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, 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 an overall topographic map. The other is called *local planning*, which relies on line of sight information, available to the robot sensors or cameras from particular perspectives. We note that human beings perform very well in path planning through a two stage anticipatory planning process [64]: 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, artificial neural nets may be adopted to implement the operations in both stages.

Global Terrain Retrieval via Pattern Association The robot navigation area can be divided into 2-dimensional or 3-dimensional 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 nets. When a robot "perceives" an environment similar to one previously trained on, the neural net uses the (continuous) values of the local grid cells to retrieve the best fitting global terrain features based on the training data. Possible candidates for this associative retrieval task are the Rumelhart's memory/learning modules (see Section 2.2) and multilayer Perceptrons (see Section 2.4). The reasons for adopting such neural nets are that they work well with continuous input values and they exhibit good fault tolerance capabilities [117,139].

Boltzmann Machine for Optimal Path Finding Once the information about the global grid cells is retrieved, then the neural net is ready to plan the optimal path to reach a ıemote goal. In order to find the global optimal path, a constrained optimization formulation can be adopted and neural nets 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 and 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 optimum, simulated annealing techniques such as a 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.

4.2.2 Path Control Using a Multilayer Perceptron

After the determination of the desired trajectory in the path planning level, the task of path control is to generate the motor control signals (torques and forces) so that the robot may be driven to follow the trajectory. Based on some physiological model [70,133], the dynamic movement of a robot or a manipulator can be controlled by a feed-forward controller. as shown in Figure 4.2. It consists of three major components: two *identical* inverse dynamics systems ($IDS_1 = IDS_2$) and the robot (or manipulator).

The IDS_1 receives the desired trajectory information $\mathbf{\Pi}_d$ and produces the corresponding desired motor command $\mathbf{T}_d$ to drive the robot so that the actual movement of the robot (denoted as $\mathbf{\Pi}$) may follow as closely as possible the desired trajectory $\mathbf{\Pi}_d$. The IDS_2, on the other hand, takes the actual movement $\mathbf{\Pi}$ and produces the reference motor command $\mathbf{T}_r$ as shown in Fig. 4.2.

To efficiently train the *IDS* to implement the feed-forward controller as shown in Fig. 4.2, a two stages learning procedure can be adopted [70,132]. The first stage is called *generalized learning* and its configuration is shown in Fig. 4.3(a).

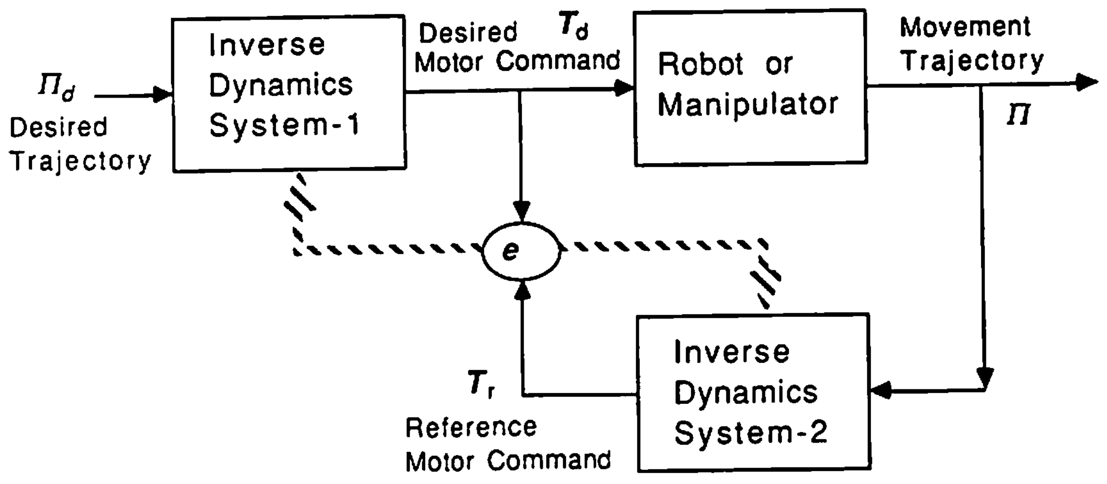


Figure 4.2: A simplified schematical diagram for the feed-forward controller.

The second stage is called ***specialized learning*** and it uses the different configuration shown in Fig. 4.3(b). In the generalized learning stage, a set of desired motor commands, denoted as $\{\mathbf{T}_d\}$. are used to drive the robot and the set of resulting trajectories is denoted as $\{\mathbf{\Pi}\}$. The *IDS* receives $\{\mathbf{\Pi}\}$ as input and yields a set of reference motor commands, denoted as $\{\mathbf{T}_r\}$. The goal of the generalized learning is to minimize the errors between $\{\mathbf{T}_r\}$ and $\{\mathbf{T}_d\}$ in the least-squares sense [133]. After the *IDS* is well trained, if a real input $\mathbf{\Pi}'$ is sufficiently close to one trajectory in the set $\{\mathbf{\Pi}\}$. the controller should be able to retrieve the proper motor commands $\hat{\mathbf{T}}$, making the actual movement $\hat{\mathbf{\Pi}}$ closely follow $\mathbf{\Pi}'$.

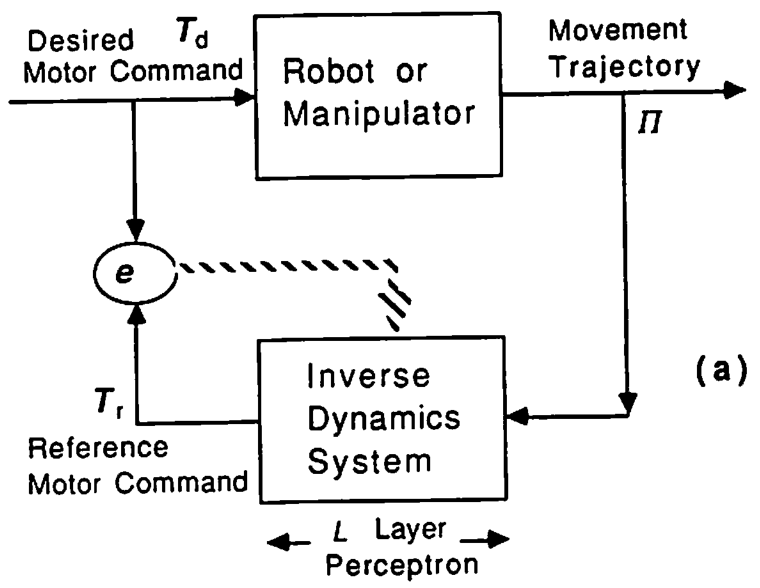
Due to the lack of knowledge about the operating range of the desired motor commands $\{\mathbf{T}_d\}$, an unnecessarily large set of $\{\mathbf{T}_d\}$ may have to be used for training. This difficulty can be overcome by incorporating a specialized learning stage into the controller system (see Fig. 4.3(b)), in which the *IDS* is trained based on the desired trajectory $\{\mathbf{\Pi}_d = [\Pi_1^{(d)}, \cdots, \Pi_m^{(d)}]\}$ and outputs the appropriate motor commands $\{\mathbf{T}\}$ to drive the robot. The actual movement trajectory of the robot is a function of $\mathbf{T}$, denoted as $\mathbf{\Pi}(\mathbf{T}) = [\Pi_1, \cdots, \Pi_m]$. 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 approach is that, in the initial step, the training of the *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. 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*. By switching back and forth between the two learning stages, the system can also avoid being trapped by local minimum [132].

Multilayer Perceptrons for Generalized Learning An L-layer neural net can be used to implement the *IDS* in a generalized learning system [133]: An input $\mathbf{T}_d = [T_1^{(d)}, \cdots, T_n^{(d)}]$ is selected and applied to the robot to obtain a corresponding $\mathbf{\Pi}$, and the network is trained to reproduce $\mathbf{T}_r = [T_1^{(r)}, T_2^{(r)}, \cdots, T_n^{(r)}]$ at its output from $\mathbf{\Pi}$ (see Figure 4.3(a)). The mathematical formulation for the updating of weights $w_{ij}(l)$ using the back propagation learning is:

$$\Delta w_{ij}(l) = \eta \, \delta_i(l) \, a_j(l-1)$$

where



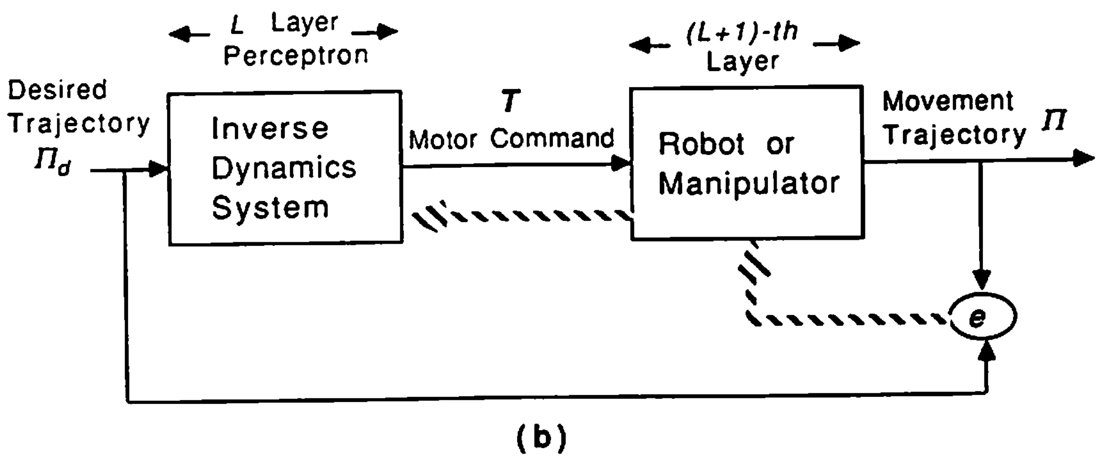


Figure 4.3: (a) Generalized learning configuration. (b) Specialized learning configuration. Note that the *IDS* can be modeled by an *L*-layer Perceptron.

$$
\begin{aligned}
\text{if } l = L, \quad \delta_i(L) &= f'(u_i(L))(T_i^{(d)} - T_i^{(r)}) \\
\text{if } l < L, \quad \delta_i(l) &= f'(u_i(l)) \sum_j \delta_j(l+1)\, w_{ji}(l+1)
\end{aligned}
$$

Note that the weight training here is based on the least squares errors of $|\mathbf{T}_d - \mathbf{T}_r|^2$, although the real objective of the robot training should have been minimizing $|\mathbf{\Pi}_d - \mathbf{\Pi}|^2$.

Multilayer Perceptrons for Specialized Learning The same L-layer neural net can be used to implement the *IDS* in a specialized learning procedure [133]: Referring to Figure 4.3(b), the dynamic model of the robot can be regarded as an additional layer, i.e., the $(L+1)$-th layer. However, BP 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 is proposed [132]:

$$
\begin{aligned}
\delta_i(L) &= f'(u_i(L)) \sum_j \delta_j(L+1) \frac{\partial \Pi_j(\mathbf{T})}{\partial T_i} \\
\delta_j(L+1) &= (\Pi_j^{(d)} - \Pi_j)
\end{aligned}
$$

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

$$
\frac{\partial \Pi_j(\mathbf{T})}{\partial T_i} \approx \frac{\Pi_j(\mathbf{T} + \Delta T_i \mathbf{I}_i) - \Pi_j(\mathbf{T})}{\Delta T_i}
$$

where $\mathbf{I}_i = [0,\ 0,\ \cdots,\ 1,\ 0,\ \cdots,\ 0]$. For the training of the remaining L layers of the multilayer Perceptron, the conventional back propagation learning algorithm (see Eqs. 2.19 and 2.20) can be adopted.

Chapter 5

Architectures for Artificial Neural Nets

This chapter advocates digital VLSI architectures for implementing a wide variety of artificial neural nets (ANNs). A programmable systolic array is proposed, which maximizes the strength of VLSI in terms of intensive and pipelined computing and yet circumvents its limitation in communication. The array is meant to be more general purpose than most other ANN architectures proposed. It may be used for a variety of algorithms in both the retrieving and learning phases of ANNs: e.g., single layer feedback nets and multilayer feed-forward nets. Although design considerations for the learning phase are somewhat more involved, our design can accommodate very well several key learning rules, such as Hebbian, delta, competitive, and back-propagation learning rules. Hardware implementation of the processing units, based on CORDIC techniques, is also discussed.

5.1 Alternatives for ANN Implementation

The hardware most suitable for implementing an ANN depends on both the application type (optimization or associative retrieval) and the intended application domain (general-purpose or special-purpose). They affect many design factors e.g., speed, learning and weight updating capabilities, system size, linear/nonlinear functionality and control circuit, I/O data link and interface, memory size, word-length, clock rate, power consumption, and so on.

This dissertation proposes digital parallel (systolic) architectures for the implementations of ANNs [93,95,90,92]. There are several existing ANN implementations worthy of consideration [48]: digital parallel architectures [162,60,3,65], analog (VLSI) electronics [54,146,4,42], and optical technologies [36,102,167,37].

5.1.1 Digital (VLSI) Parallel Implementations

Due to the parallel processing nature of the ANNs, various existing or new parallel architectures have been proposed for their implementations. As compared to the analog neural circuits, digital implementations offer higher flexibility, programmability, and precision in computation [65,162,60,3].

A single-instruction-multiple-data (SIMD) machine is a parallel array of arithmetic processors, with local connectivity between them and with local memory associated with each. Instructions are broadcast from a host, with all processors executing the same instruction simultaneously [81]. One potential SIMD candidate for ANN implementation is the *Connection machine* [49,65,162]. However, such machines are not naturally suited to ANN connectivity patterns, modifications to communication scheme are required.

Another potential ANN implementation candidate are *hypernets*, which are a type of hierarchical network that can be constructed incrementally by methodically putting together a number of basic modules [60]. Each module consists of a set of interconnected nodes. Each node has multiple ports to connect to other nodes through dedicated bidirectional links or via shared buses.

Digital electronic implementations appears to be a promising way to realize multi-layer Perceptrons. In order to implement the global interconnectivity on a locally interconnected parallel array, special routing hardware for locally interconnected multi-layer Perceptrons has been proposed to overcome the difficulty [3].

5.1.2 Analog (VLSI) Electronic Implementations

Analog VLSI technology, providing direct implementations of the differential equations of neural dynamics and exploiting the asynchronous updating properties of analog devices, can provide extremely high speed computations which are qualitatively different from those of any digital computer. Using analog computation, a neuron can be easily implemented by a differential amplifier and the synaptic weights implemented via resistors. In this way, many neurons can be "fit" into one single chip. Analog techniques have been adopted for implementing various kinds of neural nets [54,28,28,160,42,43,4,5,146].

Hopfield and Tank [54] built analog circuits, based on the continuous state neural net, to solve the traveling salesman problem, which is a difficult NP-complete problem. Although a globally optimal solution is not guaranteed, the analog circuits can reliably find the local minimum of the tour length energy surface nearest to the starting tour. The circuits have also been used to solve some other less complicated problems, e.g., the A/D converter and linear programming problems [156]. For implementing a more general purpose neural-like analog circuit, Chua and Lin developed an optimization theory for analog circuit implementation that can be used to solve a wide class of nonlinear optimization problems [28].

Analog VLSI implementation of associative memory based on a discrete state Hopfield neural net have been reported [160,42,43]. The maximum allowed capacity of one of these associative memories with N neurons is about $0.15N$ stable states. Furthermore, as the system nears maximum capacity, many spurious stable states can appear. In order to solve the limited memory capacity problem encountered in the Hopfield type neural nets, the analog circuit system is modified to be a grandmother cell circuit with differential dynamics being used in the winner-take-all mechanism [42].

By incorporating both the simulated annealing technique and a stochastic learning mechanism. an analog VLSI system based on Boltzmann machine has been implemented. Each neuron is a differential amplifier with two complementary outputs. The temperature cooling and random decision mechanisms required by the Boltzmann machine are realized by amplified thermal noise. The synaptic weights are implemented digitally by pass transistors, and the weight strength is stored as a signed R-bit number in a set of flip-flops, this implements the learning capability of the analog Boltzmann machine [4,5].

Analog VLSI has also been used to implement neural nets to perform human early vision processing, e.g., retina processing [146]. A large fraction of the retina processing is dedicated to extracting motion features (e.g., edges), which involves extensive lateral and temporal inhibition computation. Neither formulated as an optimization nor an associative retrieval problem, the pre-processing conducted by our retinal ganglion cells can be efficiently implemented by locally interconnected analog VLSI neural circuits [146].

5.1.3 Optical (Electro-Optical) Implementations

Optical computing approaches, exploiting the global interconnectivity of optical signal flow, have been proposed for the implementations of ANNs [36,113,35]. No virtual optical ANN system (purely optical PEs and synaptic weights that can be time multiplexed) has been built yet. Systems of this type may someday become the most important class of ANN implementations due to the capabilities of massive optical storage, global interconnectivity, and high speed processing [48].

An optical implementation of content addressable associative memory based on the discrete state Hopfield neural net has been proposed [36]. This optical implementation performs the associative recall (a matrix-vector multiplication operation) by means of outer-product technique. The synaptic weights are implemented as a photographic transparency (i.e., a mask), and the neuron output are sent into and pass through the weight mask (the Bragg cells) via parallel planar laser beams. The weighted sums are collected by converging the attenuated beams onto a linear detector array. The film optical amplifiers are then used to perform the threshold neuron activations. This implementation also provide useful links between optical neural processing and pattern recognition/machine vision.

Due to the low memory capacity in the Hopfield associative memory model, the weight matrix formed by the outer product method has low rank which can be exploited to reduce the computation. An inner product technique to perform the matrix-vector multiplication has also been proposed [102]. This technique achieves a considerable amount of hardware savings. More significantly, real-time inputting and updating of the synaptic weights can be potentially implemented with existing space-variant holographic elements and recently discovered liquid crystal television spatial light modulators.

A modular electro-optical ANN system capable of pattern learning has been proposed [37]. Each module is a complete associative memory which adapts as it is exposed to associated information patterns. All these modules can be optically cascadable, with all inputs and outputs in the form of 1-D or 2-D image beams or intensity arrays. Various learning modules have been proposed, e.g., delta learning, Hebbian learning, and differential learning.

A new approach to learning in multilayer optical neural nets based on holographically interconnected nonlinear devices has also been proposed [167]. This optical net uses self-aligning volume holograms to bidirectionally interconnect nonlinear etalons which act as the bidirectional optical neurons. This architecture combines the robustness of the distributed neural computation and the back-propagation learning procedure with the high speed processing of nonlinear etalons, the self-aligning ability of phase conjugate mirrors, and the massive storage capacity of volume holograms to produce a powerful and flexible parallel optical ANN.

5.2 Mapping Algorithms to Systolic/Wavefront Arrays

The major emphasis of VLSI system design is to reduce the overall interconnection complexity and to keep the overall architecture highly regular, parallel, and pipelined. It stresses the importance of local communication in the array processor. *Systolic* and *wavefront* arrays are a new class of pipelined array architectures, well suited to VLSI implementation, because of their properties of modularity, regularity, local interconnection, and pipelining. A systolic array is a network of processors which rhythmically compute and pass data through the system [80,79]. In contrast, a wavefront array is an asynchronous, self-timed, data-driven computation array, which features high regularity, modularity, and local interconnection [91,89,86].

The main concern in algorithm-oriented array processor design is: *given an algorithm, how is an array processor systematically derived?* In the following, a systematic mapping methodology for deriving systolic/wavefront arrays for ANNs is introduced. Our design begins with a (data) dependence graph (DG) to express the recurrence and parallelism. Next, this description will be mapped onto a systolic/wavefront array [81,110,84]. Although in the later part of this dissertation we only concentrate on mapping algorithms to systolic arrays, similar methods can be used to map algorithms onto wavefront arrays [81].

5.2.1 Deriving DGs from Given Algorithms

A DG is a directed graph which specifies the data dependencies of an algorithm. In a DG, *nodes* represent computations, and *arcs* specify the data dependencies between computations. In our notation, a dependence arc's terminating node depends on its initiating node. From an initial sequential algorithm description, a DG for the algorithm can be derived by first converting the algorithm to a *single assignment form*, in which the value of any variable is assig..d only once in the algorithm [81]. A single assignment form can clearly show the data dependencies in the algorithm. For regular and recursive algorithms, the DGs will also be regular and can be represented by a grid model; therefore, the nodes can be specified by simple indices, such as (i, j, k). *Design of a locally linked DG is a critical step in the design of systolic arrays.*

5.2.2 Mapping DGs onto Systolic Arrays

Two tasks are involved in mapping a DG onto a systolic array. The first task is *processor assignment* and the second is *schedule assignment* [81]. It is common to use a *linear projection* for processor assignment, in which nodes of the DG along a straight line are projected (assigned) to a PE in the processor array (see Figure 5.1(a)). A *linear scheduling* is also common for schedule assignment, in which nodes on a hyperplane in the DG are scheduled to be processed at the same time step (see Figure 5.1(b)).

Processor Assignment via Linear Projection

Mathematically, a linear projection is often represented by a ***projection vector*** $\vec{d}$. Since the DG of a locally recursive algorithm is very regular, the linear projection maps an n-dimensional DG onto an $(n-1)$-dimensional lattice of points, known as the *processor space*. As an example, the 2-D index space of the DG shown in Figure 5.1 may be decomposed into a direct sum of a 1-D *processor space* and 1-D *delay space*. The delay space is related to the scheduling as explained below.

Schedule Assignment via Linear Scheduling

A *scheduling* scheme specifies the sequence of the operations in *all* PEs. More precisely, a schedule function represents a mapping from the n-dimensional index space of the DG onto a 1-D schedule (time) space. A linear schedule is based on a set of parallel and uniformly spaced hyperplanes in the DG. These hyperplanes are called *equitemporal hyperplanes*—all the nodes on the same hyperplane must be processed at the same time. A linear schedule can also be represented by a *schedule vector* $\vec{s}$, which points in the direction normal to the hyperplanes. For any index point $\mathbf{i}$ in the DG, its time step is $\vec{s}^T\mathbf{i}$. A set of (linear schedule) hyperplanes and their associated schedule vector are illustrated in Figure 5.1(b).

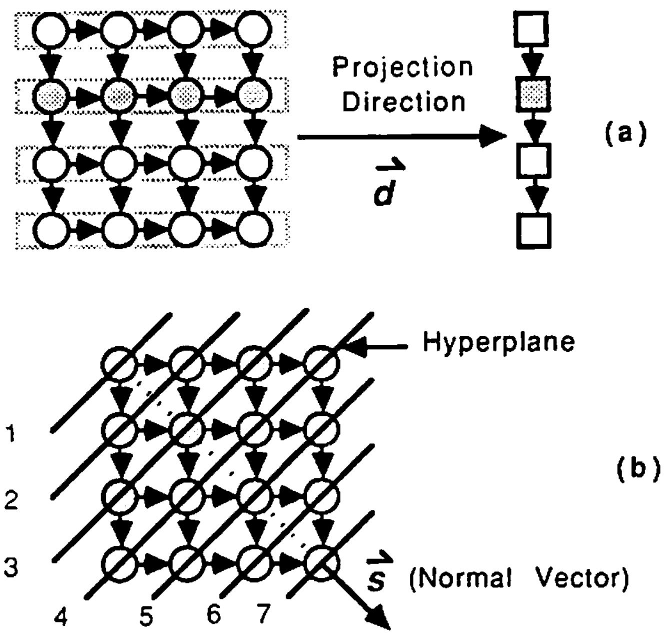


Figure 5.1: (a) A linear projection with projection vector $\vec{d}$; (b) A linear schedule $\vec{s}$ and its hyperplanes.

Systolic Schedules

Given a DG and a projection direction $\vec{d}$, not all schedule vectors $\{\vec{s}\}$ qualify to define a valid schedule for the DG. Some of them may violate the precedence relations specified by the dependence arcs. For systolic design, the schedule vector $\vec{s}$ in the projection procedure must satisfy the following conditions:

$$\vec{s}^T \vec{e} > 0 \tag{5.1}$$

$$\vec{s}^T \vec{d} \neq 0 \tag{5.2}$$

where $\vec{e}$ is any dependence vector in the DG. Equation 5.1 means that every edge of the resulting systolic array will have one or more delay elements, which satisfies the *temporal locality* condition in the definition of the systolic array [81]. Equation 5.2 implies that nodes on an equitemporal hyperplane should *not* be projected to the same PE. The permissible hyperplane directions defined by the first condition are the same as the notion of the time cone in [69,32].

5.2.3 Design Example: Matrix-Vector Multiplication

Let us consider the systolic array design for the matrix-vector multiplication problem, i.e.,

$$\mathbf{c} = \mathbf{A}\ \mathbf{b}$$

The sequential FORTRAN code for this matrix-vector multiplication problem can be written as

```
      do 10 i = 1,4
      c(i) = 0.
      do 10 j = 1,4
      c(i) = c(i) + a(i,j) * b(j)
10    continue
```

Note that in this program, $c(i)$ is overwritten many times to save storage space, thus the value of $c(i)$ is assigned more than once. To transform the above program to a *single assignment* code, where every variable is only assigned once during the execution of the algorithm, the number of indices of the vector **c** should be increased. The corresponding *localized* single assignment code, where the change of indices is incremental, can be written by introducing the the intermediate variables $a(i,j)$, $b'(i,j)$, and $c'(i,j)$:

```
      do 10 $i = 1,4$
      $c'(i,1) = 0.$
      do 10 $j = 1,4$
      $b'(i+1,j) = b'(i,j)$
      $c'(i,j+1) = c'(i,j) + a(i,j) * b'(i,j)$
10    continue
```

where the initial values are $\{b'(1,j) = b(j)\}$, and the final results $\{c(i) = c'(i,5)\}$. Note that each element of the matrix $\{c'(i,j)\}$ is assigned once only.

By viewing each dependence relation as an arc between the corresponding variables located in the index space, the DG shown in Figure 5.2(a) can be obtained. In this DG, the operation at each node is specified in Figure 5.2(b). If we select the projection vector to be $\vec{d} = [1\ 0]$ and the schedule vector to be $\vec{s} = [1\ 1]$, we can get a linear systolic array in which all the inputs $\{b_i\}$ originally reside in each PE, and the outputs $\{c_i\}$ are pipelined out one-by-one from the boundary PEs (see Figure 5.2(c)). On the other hand, if we select the projection vector to be $\vec{d} = [0\ 1]$ and the schedule vector to be $\vec{s} = [1\ 1]$, we can get another linear systolic array with all the inputs $\{b_i\}$ pipelined one-by-one into the array, and the outputs $\{c_i\}$ remaining in each PE (see Figure 5.2(d)). It is also possible to project the DG in the diagonal direction, which can lead to hardware saving when the **A** matrix is banded. Among these many alternatives, optimization criteria with constraints can be imposed to choose the most suitable systolic arrays for specific applications.

This mapping methodology has been adopted to derive systolic/wavefront array architectures for various numerical, signal/image processing, and neural nets algorithms [83]. Successful design examples in numerical and signal/image applications include: transitive closure and the shortest path problems [85,87], Kalman filtering for signal tracking [97,98,99,94], multichannel least squares lattice filter [106,105], dynamic time warping for speech recognition [82], stochastic relaxation for image restoration [82,100], rank order filtering for image enhancement [100], Hough transform for shape analysis [88], 2-D correlation for pattern matching [100], and 2-D rank decomposition for mask processing [100].

5.3 Systolic Architectures for Artificial Neural Nets

We shall demonstrate that a simple programmable architecture can be developed to accommodate most of the neural processing algorithms.

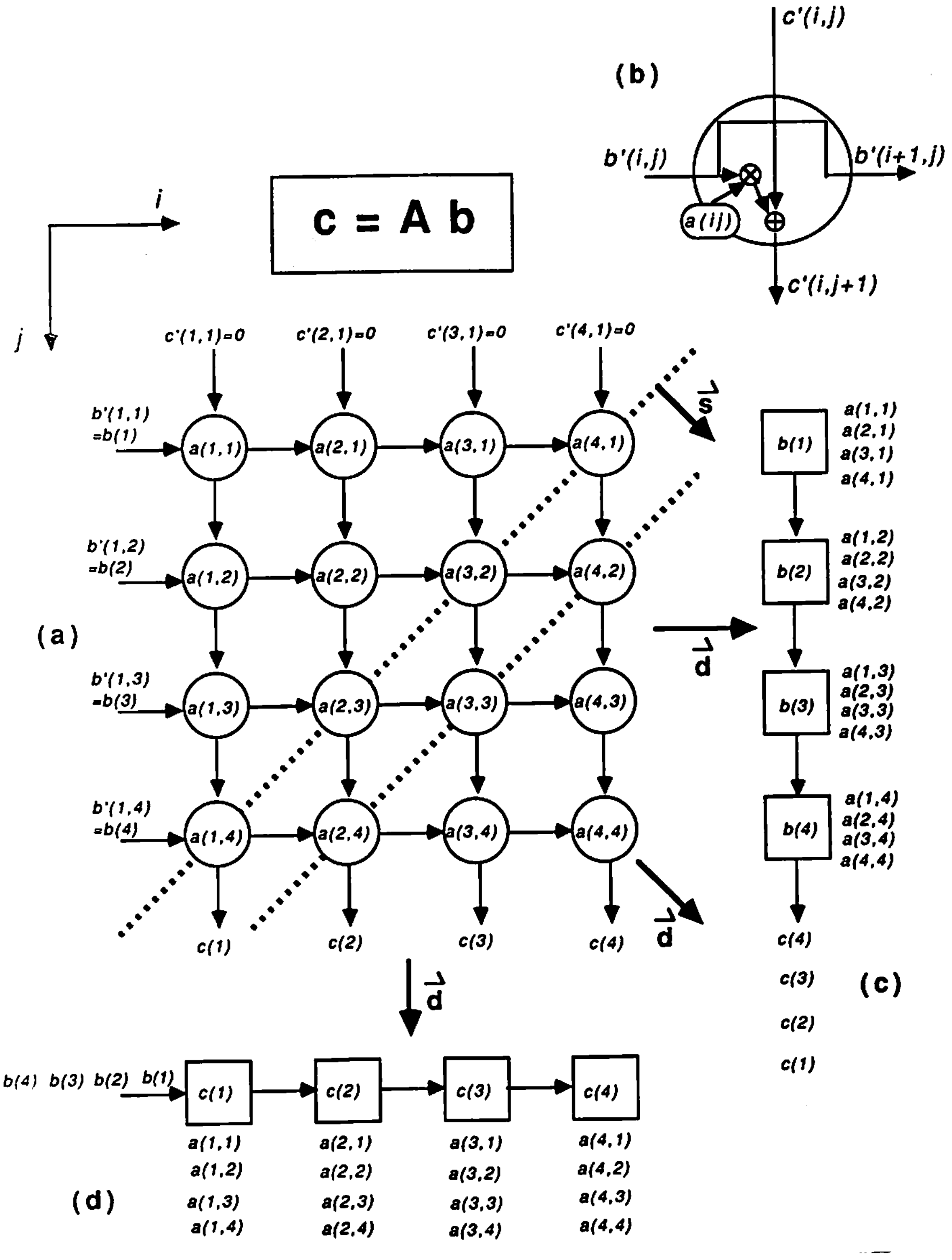


Figure 5.2: (a) Localized DG for matrix-vector multiplication. (b) The functional operation at each node of the DG. (c) A systolic array obtained by selecting $\vec{d} = [1\ 0]$ and $\vec{s} = [1\ 1]$. (d) A systolic array obtained by selecting $\vec{d} = [0\ 1]$ and $\vec{s} = [1\ 1]$.

5.3.1 Systolic Design for Single-Layer ANNs

Ring Systolic ANN For Retrieving Phase

The system dynamics of the retrieving phase of single layer feedback neural nets (e.g., Hopfield neural nets, memory/learning modules, and Boltzmann machine) can all be formulated as a unified consecutive matrix-vector multiplication problem interleaved with the nonlinear activation function shown below [99,95,96]. (Here, parallel updating of Hopfield neural nets and Boltzmann machines are assumed.)

$$\begin{aligned} \mathbf{u}(k+1) &= \mathbf{W}\mathbf{a}(k) + \theta \\ \mathbf{a}(k+1) &= F[\mathbf{u}(k+1), \mathbf{u}(k), \mathbf{a}(k)] \end{aligned} \tag{5.3}$$

where the $F[\mathbf{x}]$ operator performs the nonlinear activation function f. The vectors and matrices used are:

$$\begin{aligned} \mathbf{u} &= [u_1, u_2, \cdots, u_N]^T \\ \mathbf{a} &= [a_1, a_2, \cdots, a_N]^T \\ \theta &= [\theta_1, \theta_2, \cdots, \theta_N]^T \end{aligned}$$

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{N1} & w_{N2} & \cdots & w_{NN} \end{bmatrix} \tag{5.4}$$

DG Design for Consecutive Matrix Vector Multiplications The consecutive matrix-vector multiplication array architecture design can be derived from a cascaded DG [81]. If the conventional DG design for matrix-vector multiplication is used (see Figure 5.2). it will lead to a cascaded DG. Implementation of this DG by a linear array requires nonlinear processor assignment as shown in Figure 5.3(a) (with linear schedule along diagonal direction), the result is a locally interconnected systolic array with bidirectional communication links (see Figure 5.3(b)). This systolic architecture requires smart switches to change the operations of each PE at different time slots. Moreover, some synaptic weights $\{w_{ij}\}$ need to be simultaneously stored in several PEs. This makes the design for adaptive modification of $\{w_{ij}\}$ in the learning phase very difficult.

The problem can be overcome by a proper modification on the DG: The data ordering of the $\{w_{ij}\}$ elements can be rearranged so that the direction of the inputs $\{a_i(k)\}$ in the DG becomes aligned with that of the outputs $\{a_i(k+1)\}$ [81], as depicted in Figure 5.4(a). In this DG. the i-th row of the $\mathbf{W}$ matrix is placed on the i-th column of the $\{w_{ij}\}$ data array in DG, and for $i = 1, 2, \ldots, N$, the i-th column of the $\{w_{ij}\}$ data array is circularly-shifted-up by $i-1$ positions. Since the input (from the top) and the output (at the bottom) are now aligned in the same direction. then many copies of such DGs can be cascaded top-down, with the input and output data perfectly interfaced. Based on this new cascaded DG, the mapping from DG to systolic array is straightforward. The projection can be taken along the vertical direction, i.e., $\vec{d} = [i,\ j] = [0,\ 1]$, and the schedule hyperplanes are chosen to be along the horizontal direction, i.e., $\vec{s} = [i,\ j] =$

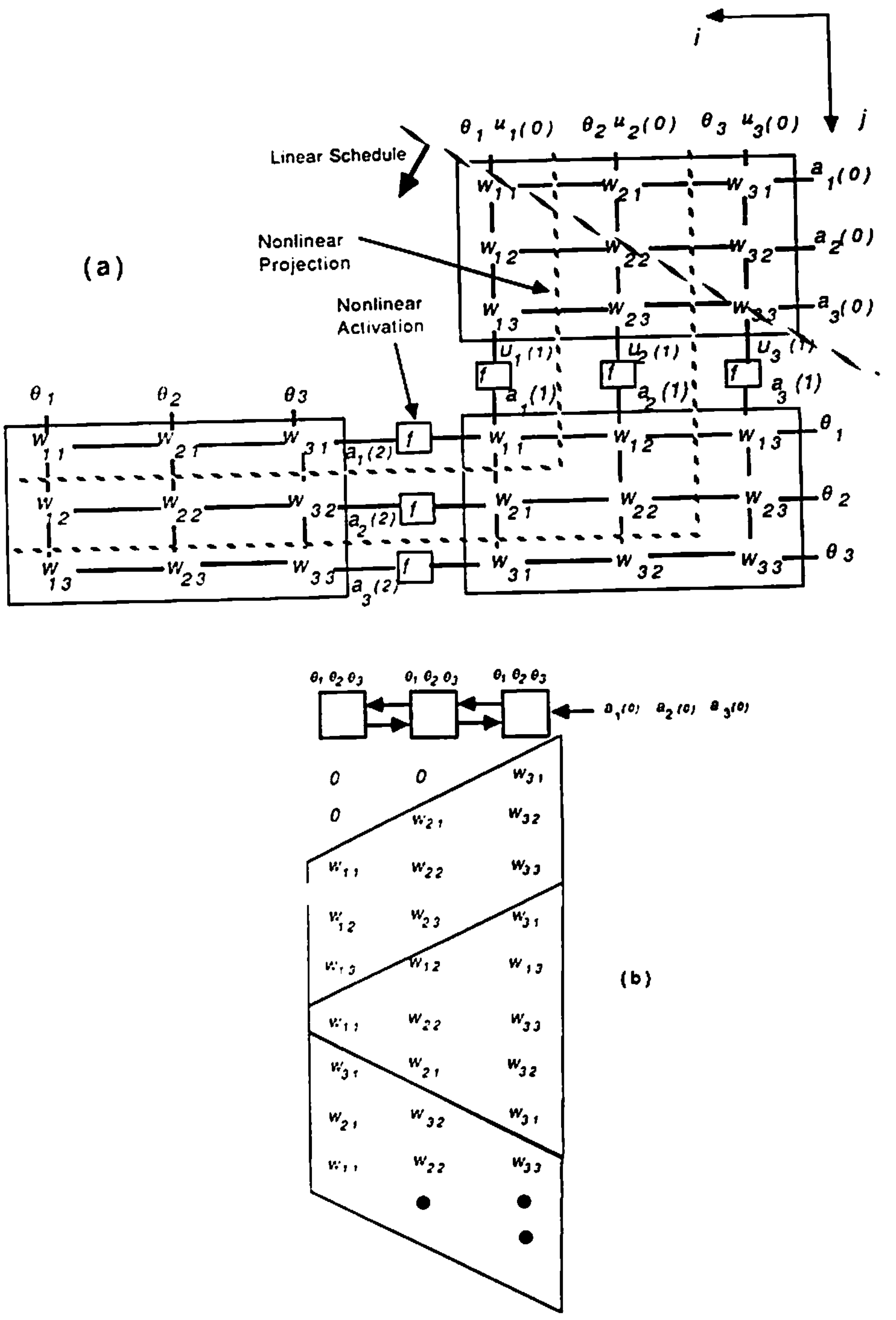


Figure 5.3: (a) Cascaded DG for consecutive matrix-vector multiplication formulation using conventional approach. (b) The corresponding systolic array.

[0, 1]. This leads to a ring systolic architecture as shown in Figure 5.4(b). The pipelining period of this design is 1, which implies 100% utilization efficiency [81]. A small disadvantage of the design is that the (global) spiral communication link is required.

Operations in the Ring Systolic ANN In the retrieving phase, each PE (say the j-th PE) can be treated as a neuron, and the corresponding synaptic weights $(w_{j1}, w_{j2}, \ldots, w_{jN})$ are stored in the memory of the j-th PE in a circularly-shift-up order. The operations at the $(k+1)$-th iteration can be described as follows:

1. Each of the neuron activation outputs, $a_j(k)$, will move counter-clockwise across the ring array and visit each of the other PEs once in N clock units.
2. When $a_j(k)$ arrives at the i-th PE, it is multiplied with w_{ij}, and the p. duct is added to the θ_i to derive $u_i(k+1)$ (see Eq. 1.1). After N clock units, all the products are collected and the resulting $u_i(k+1)$ is ready for the nonlinear activation operation.
3. The newly activated neuron output $a_i(k+1)$ is propagated to the left-side neighbor PE.

The above procedure can be executed in a pipelined fashion for all $j \in [1, N]$ and $i \in [1, N]$. Moreover, it can be recursively executed (with increased k) until the system convergence is reached.

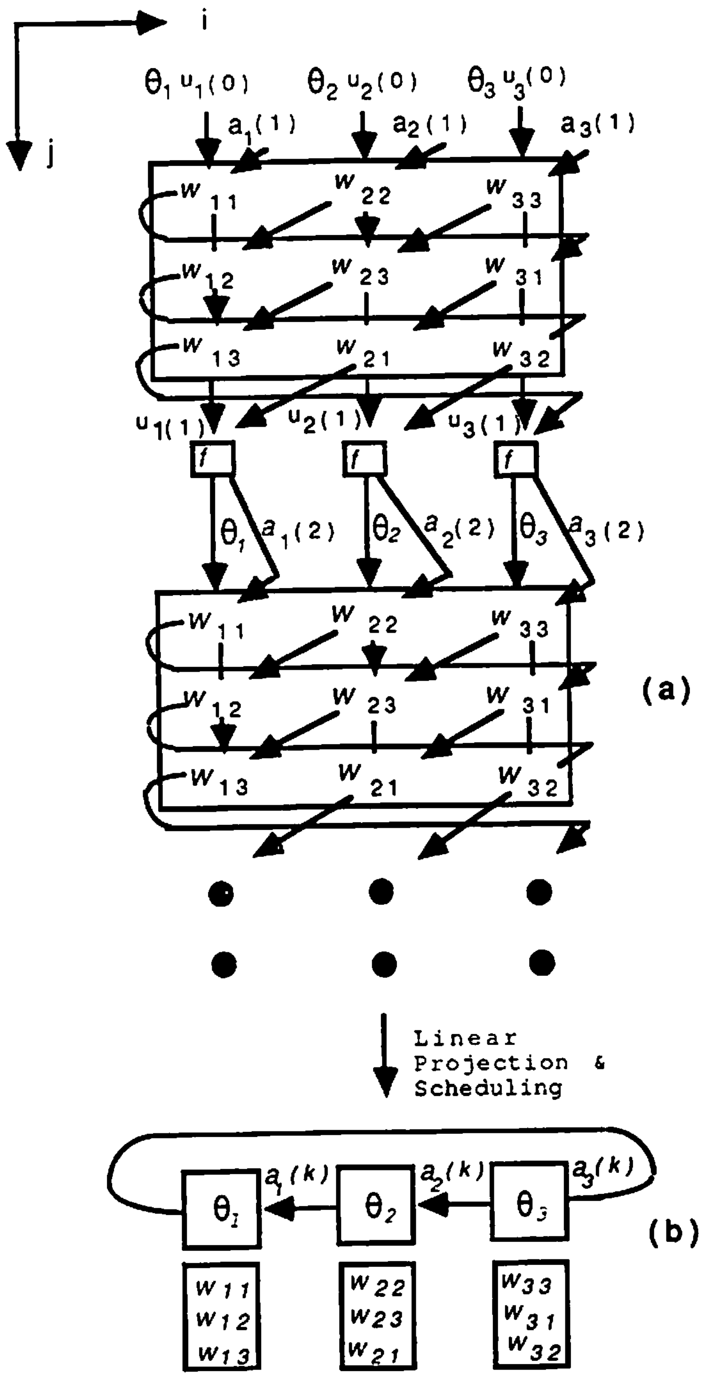

i
j
$\theta_1\ u_1(0)$
$\theta_2\ u_2(0)$
$\theta_3\ u_3(0)$
$a_1(1)$
$a_2(1)$
$a_3(1)$
w_{11} w_{22} w_{33}
w_{12} w_{23} w_{31}
w_{13} w_{21} w_{32}
$u_1(1)$
$u_2(1)$
$u_3(1)$
f
θ_1 $a_1(2)$
θ_2 $a_2(2)$
θ_3 $a_3(2)$
(a)
Linear Projection & Scheduling
$a_1(k)$
$a_2(k)$
$a_3(k)$
θ_1 θ_2 θ_3
w_{11} w_{12} w_{13}
w_{22} w_{23} w_{21}
w_{33} w_{31} w_{32}
(b)

Figure 5.4: (a) Cascaded DGs for consecutive matrix-vector multiplication formulation. (b) The corresponding ring systolic array.

Ring Systolic ANN for the Learning Phase

The systolic array design for the retrieving phase can be adapted to apply to the learning phase [90,96]. Most learning rules for single layer feedback neural nets. such as the Hebbian learning rule [47], delta learning rule [172], Boltzmann learning rule [1]. and competitive learning rule [44]. can be implemented quite efficiently on the ring systolic ANN. In the following. only the cases for delta learning and Hebbian learning are discussed.[1]

Suppose that the system convergence is reached after K iterations of retrieving operations upon the presentation of the m-th training pattern, and the final activation value produced at the i-th PE is $a_i^m(K)$. (For convenience, we shall denote $a_i^m = a_i^m(K)$.) Then the next iteration (N clocks) will be devoted to the adjusting of the weights, the key task of the learning phase.

Systolic ANN for Delta Learning The delta learning rule is typically applied to a set of pairs of patterns. each pair consisting of an input pattern and a target output pattern. The goal is that when an input pattern is presented, the corresponding output pattern will be retrieved. This learning rule uses the following formulation:

$$\triangle w_{ij}^{(m)} = \eta \, (t_i^{(m)} - a_i^{(m)}) \, a_j^{(m)} \tag{5.5}$$

where $\Delta w_{ij}^{(m)} \propto -\partial\zeta^{(m)}/\partial w_{ij}^{(m)}$, and the error measurement function $\zeta^{(m)}$ is defined as

[1]The Boltzmann learning rule is basically a two steps Hebbian rule, so the implementation follows that of the Hebbian rule.

$$\zeta^{(m)} = \sum_{i=1}^{N}(t_i^{(m)} - a_i^{(m)}(K))^2$$

The operations of the delta learning rule can be written as an outer product formulation:

$$\mathbf{W}^{(m+1)} = \mathbf{W}^{(m)} + \eta\delta^{(m)}\ \mathbf{a}^{(m)^T}$$

The DG for outer product operations shown in Figure 5.5 has the same configuration as that of matrix-vector multiplication (see Figure 5.2), so it can be directly modified into a spiral DG structure, and the ring systolic array can thus be adapted for this learning rule. The operations can be briefly described as follows:

1. The value of $\eta\ (t_i^{(m)} - a_i^{(m)})$ is stored in the i-th PE. The activation value $a_j^{(m)}$ produced at the j-th PE will be cyclically propagated to all other PEs in the ring systolic ANN during the N clocks.

2. When $a_j^{(m)}$ arrives at the i-th PE, it is multiplied with the stored value $\eta\ (t_i^{(m)} - a_i^{(m)})$ to yield $\Delta w_{ij}^{(m)}$, which will then be used to produce the new $w_{ij}^{(m+1)}$ based on Eq. 1.4. Note that the data $\{w_{ij}^{(m)}\}$ are retrieved in a circularly-shift-up sequence, just like that used in the retrieving phase.

3. After N clocks, all the $N \times N$ new weights $\{w_{ij}^{(m+1)}\}$ are generated. The ring systolic ANN is now ready for the learning phase for the next training pattern.

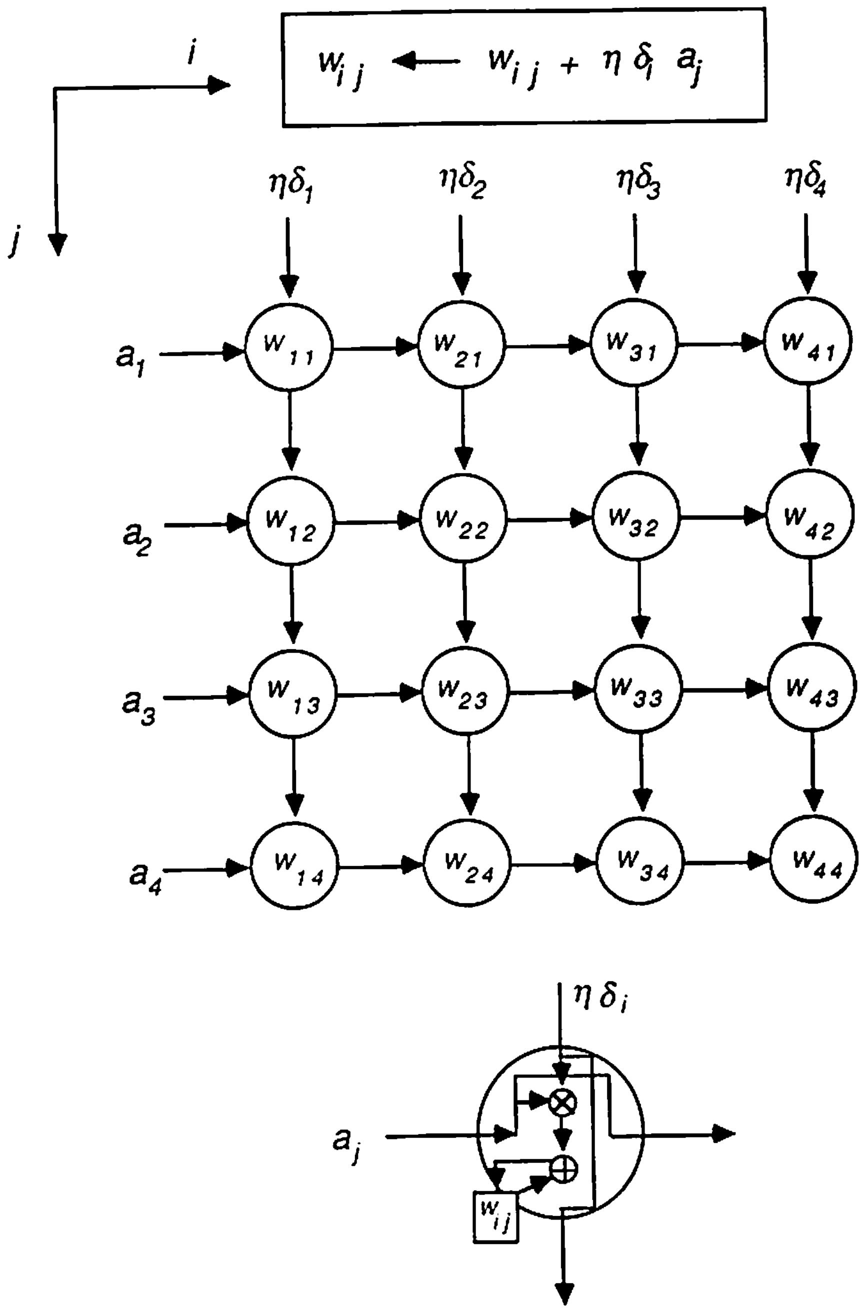


Figure 5.5: The DG for outer product operations in the delta learning rule.

Systolic ANN for Hebbian Learning The Hopfield neural nets used a simplified Hebbian learning rule for weight determination, where a synaptic weight $w_{ij}^{(m)}$ will be strengthened if the activation values of both the i-th and the j-th neurons are high [140]. This form of the Hebbian learning rule follows the simple formulation (see [140]):

$$\triangle\, w_{ij}^{(m)} = \eta\ a_i^{(m)}\ a_j^{(m)} \tag{5.6}$$

It can be easily shown that the systolic implementation of this Hebbian learning rule can be done in almost the same manner as the delta learning, except that the weight update in Eq. 5.6 (instead of Eq. 5.5) should be used for the Hebbian learning.

5.3.4 Cascaded Systolic ANN for Multilayer Perceptron

Retrieving Phase

As we discussed in Section 2.4, the mathematical formulation for the multilayer Perceptron basically follows the single layer feedback case, except that the iteration time is now replaced by the layer number. To maximize the parallelism of the operations, we can assign one ring systolic array (derived in Section 5.3.1) to each of the layers. For the convenience of architectural design, it is assumed that all the layers in the multilayer Perceptron have a uniform size of neural units.[2]

[2]One simple technique to force equal number of hidden units per layer is to artificially create a certain number of "no-op" neural units (i.e., the weights connected to the units are set to be zero) to balance any inequality between two layers.

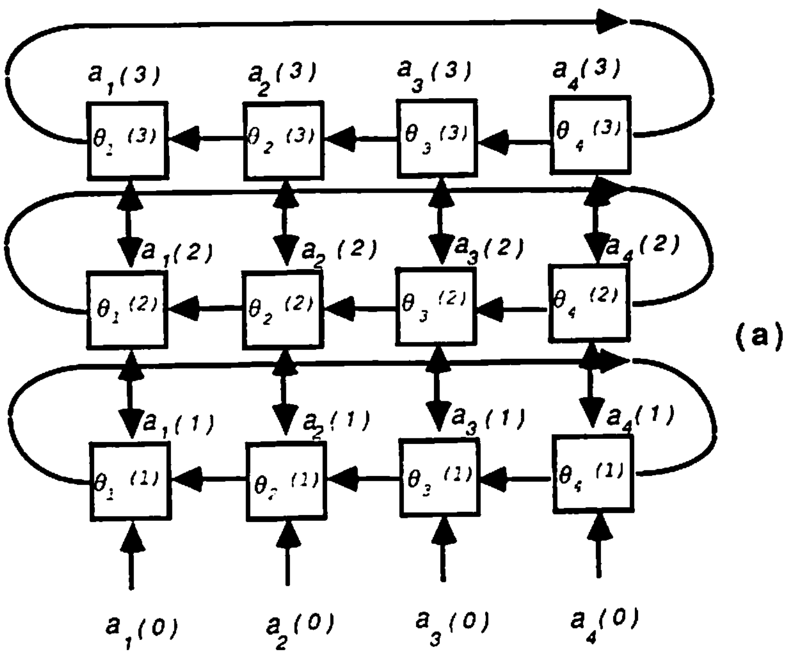
With this assumption, all the L ring arrays can now be perfectly cascaded to form a mesh array as shown in Figure 5.6(a) [90,96].

Learning Phase

The learning phase of a multilayer Perceptron adopts the back propagation learning rule, which involves two steps: the forward step and the backward step (see Section 2.4). The systolic design for the forward step is the same as that in the retrieving phase, but for the backward step it is somewhat more complicated(see Eq. 2.19 and Eq. 2.20). The implementation of Eq. 2.19 for the L-th stage of the cascaded systolic ANN is almost the same as that of the delta learning discussed in Section 5.3.1, except for the additional multiplication of the derivative term f'. The cascaded systolic design of Eq. 2.20 is briefly described as follows: Since Eq. 2.20 is basically a vector-matrix multiplication operation of $\delta_j^{(m)}(l+1)$ and $\{w_{ji}^{(m)}(l+1)\}$, the cascaded ring systolic array can naturally be applied to these operations. Assume that the error signal $\{\delta_j^{(m)}(l+1)\}$ and the weight $w_{ji}^{(m)}(l+1)$ are now available at j-th PE of the $(l+1)$-th layer:

1. The weight value $w_{ji}^{(m)}(l+1)$ is used for the synaptic updating of Eq.2.18 to obtain $\{w_{ji}^{(m+1)}(l+1)\}$ at $(l+1)$-th layer.

2. Simultaneously, the calculations of $\{\delta_i^{(m)}(l)\}$ can be started. The value $w_{ji}^{(m)}(l+1)$ is also used to multiply the residing error signal $\delta_j^{(m)}(l+1)$ (see Figure 5.6(b)). The product is added to the newly arrived parameter $acc_i^{(m)}(l)$. (The parameter $acc_i^{(m)}(l)$ is initiated at the i-th PE with zero initial value and circularly shifted leftward across the ring array.)



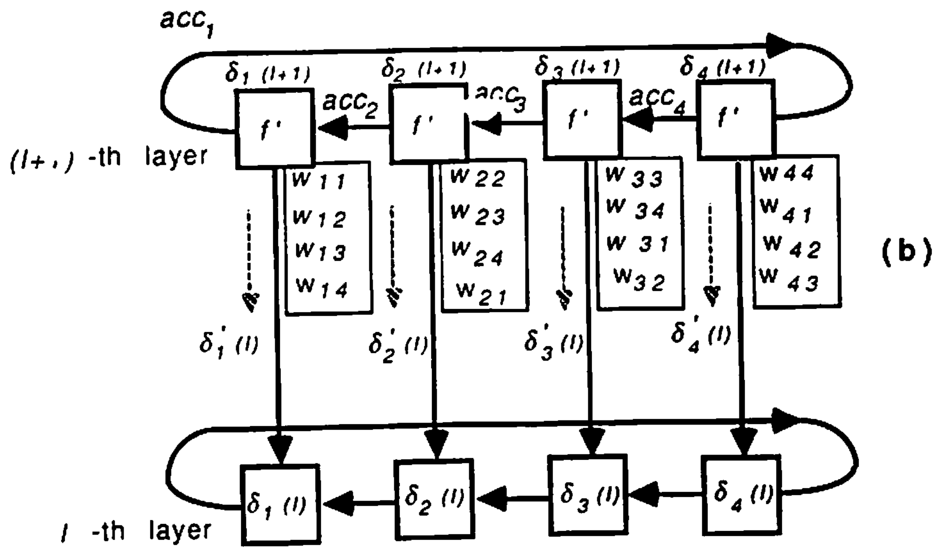


Figure 5.6: (a) A cascaded systolic ANN for the retrieving phase of multilayer Perceptrons. (b) A cascaded systolic ANN for the learning phase of multilayer Perceptrons.

3. Repeat the same operation in *Step 2* for different i, (where $i = j, j+1, \ldots, N, 1, \ldots, j-1$). After N such operations, the parameter $acc_i^{(m)}(l)$ will return to i-th PE after accumulating all the products $\delta_i'(l) = \sum_j \delta_j^{(m)}(l+1)\, w_{ji}^{(m)}(l+1)$ shown in Eq. 2.20 (see Figure 5.6(b)). This value is immediately back-propagated to the i-th PE at the l-th layer, where it will be multiplied with $f'(u_i^{(m)}(l))$ to yield the new error signal $\delta_i^{(m)}(l)$. (The value $f'(u_i^{(m)}(l))$ has been previously calculated and stored in the i-th PE of the l-th layer.)

4. The error signal $\delta_i^{(m)}(l)$ can now be used for the weight updating of $\{w_{ij}^{(m)}(l), j = i, i+1, \ldots, N, 1, \ldots, i-1\}$, and also for the calculations of $\{\delta_i^{(m)}(l-1)\}$ at the l-th layer.

Repeat *Steps* 1 - 4 for the l-th layer.

Systolic Design for Multilayers with Different Sizes

The cascaded systolic design proposed above is suitable when the sizes of the different layers are approximately equal. If the hidden unit layer has a much larger size than the I/O layers, then the design is no longer suitable and a new design may be considered: Let us use an example of a two layer neural net with configuration (3-9-4). The mappings of the operations for the retrieving/learning in the output and hidden layers are shown in Figure 5.7(a) and (b) respectively. In each mapping, there is a DG for the retrieving phase and a DG for the learning phase. For an efficient utilization of hardware, the projection direction is chosen so that the minimum number of PEs will be needed. As shown in Figure 5.7(a), a vertical projection direction is used; on the other hand, a horizontal direction is used in Figure 5.7(b). Note that in Figure 5.7(b), the DG plane for the learning phase should be stacked on top of the DG plane for the retrieving phase before the projection takes place. This will again lead to a ring systolic design as shown in Figure 5.7(c).

Design Consideration for the Retrieving Phase As shown in Figure 5.7(c), the retrieving phase of this system starts by loading the inputs $\{a_j(0),\ j = 1, 3\}$ so that they reside in the corresponding PEs in the first layer. The external input values $\{\theta_i(1),\ i = 1, 9\}$ are pipelined one-by-one into the array from the leftmost PE. Now Eq. 2.16 can be performed by creating a token at this PE with initial value $\theta_i(1)$ and propagating it along the array to accumulate all the products $w_{ij}(1)a_j(0)$ at the j-th PE, $j = 1, \ldots, 3$. The resulting net innut $u_i(1)$ will appear at the rightmost PE and the activation function can be performed to derive a new activation output $\{a_i(1),\ i = 1, \ldots, 9\}$ of the hidden layer. (Note that the derivatives $\{f'(u_i(1))\}$ can be computed now, they will be useful for the learning phase.)

The data $\{a_i(1)\}$ are sequentially pumped into the PEs of the output layer from the right-hand-side. When $a_i(1)$ arrives at the k-th PE, it is multiplied with $w_{ik}(2)$, and the product is added to the residing $\theta_k(2)$ at the k-th PE. After all nine data $\{a_i(1)\}$ pass the k-th PE, $u_k(2)$ is produced. Again it is ready for the activation operation and $a_k(2)$ will be output from the k-th PE. Like the cascaded ring systolic ANN, such a design can also deliver fully pipelined

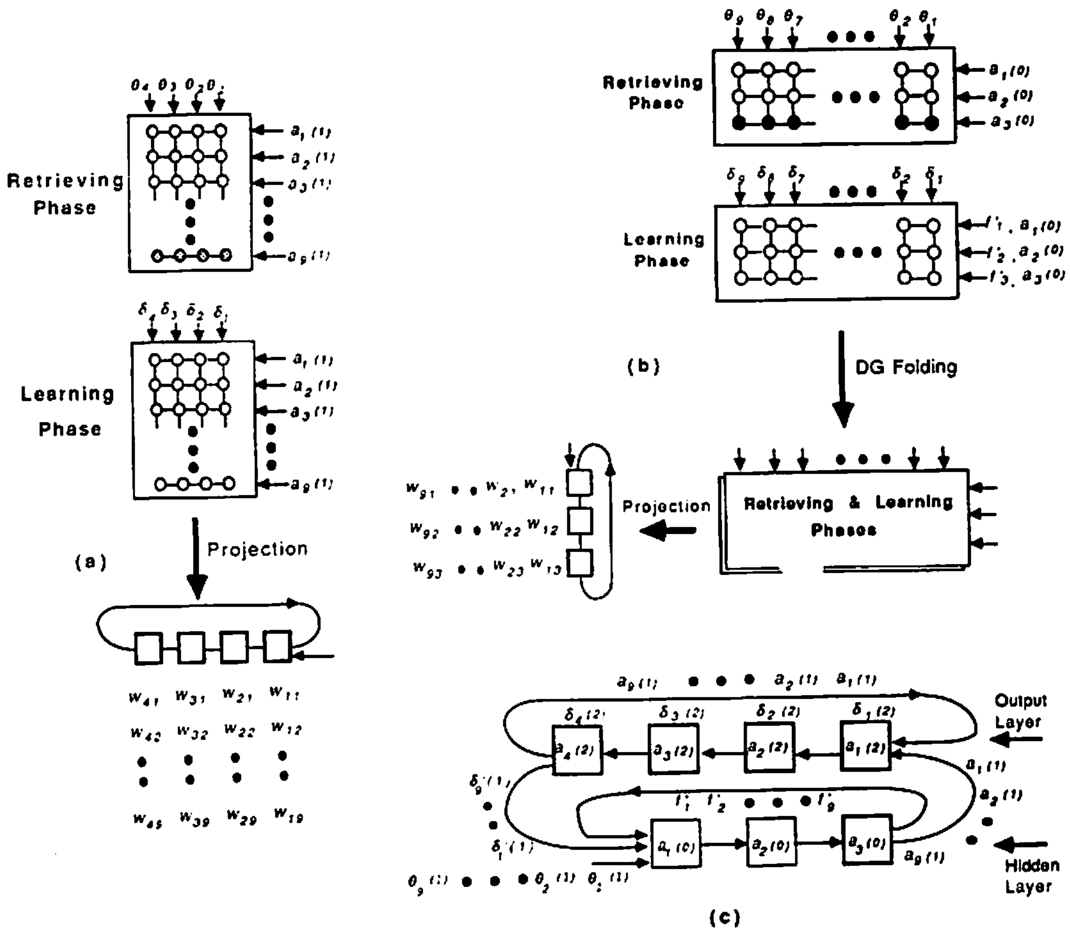


Figure 5.7: Systolic design for a multilayer Perceptron with 3-9-4 configuration. (a) The DG and array designs for the retrieving/learning of the output layer. (b) The DG and processor designs for the retrieving/learning of the hidden layer. (c) The overall systolic array design.

capability (i.e., 100% utilization efficiency) in the retrieving phase. Although the control is admittedly more complicated in this version of the systolic design, the overall hardware saving is worthwhile when the number of units at each layer are considerably different.

Design Considerations for the Learning Phase Recall that, in the back-propagation learning, the forward step should be executed before the actual weight updating in the backward step. The operations in the forward step follow exactly that of the retrieving phase. Whenever the activation value $a_k(2)$ at the k-th PE is created in the forward step, the backward step can be started. The difference between the activation value and the target value t_k is multiplied by f' to derive the error signal $\delta_k(2)$ at the k-th PE.

To compute the error signal of the hidden layer (i.e., $\delta_k(1)$) according to Eq. 2.20, a token is created at the rightmost PE (with zero initial value) and propagated along the upper array to accumulate all the products $\delta_k(2)w_{ki}(2)$ at k-th PE, $k = 1,4$ in a pipelined fashion. The final results $\{\delta_i'(1)\}$ will be available at the leftmost PE of the upper array. They will be sent sequentially to the leftmost PE at the lower array (for the hidden layer), where they will be multiplied by the derivatives $\{f'(u_i(1))\}$ to yield the new error signal $\{\delta_i(1)\}$. (The data $\{f'(u_i(1))\}$, previously computed in forward step, may be recycled for the current usage by a wrap-around link (with FIFO buffering).)

The weight updating of the output layer can be performed at the same time when the error signals are being processed. The weight value $w_{ki}(2)$, residing in the k-th PE. can be updated based on Eq. 2.18. Note that the updating again makes use of the data $\{a_i(1)\}$ previously used for computing $\{u_k(2)\}$ in the forward step. Therefore, a wrap-around link (with a FIFO for buffering) is again useful for the recycling of these data.

Now the weight updating for the hidden layer, based on Eq. 2.18, can be performed in a similar fashion. Upon the derivation of the first $\delta_1(1)$, and the weight updating of $w_{11}(1)$, the system is ready for the learning phase of the next pair of training patterns.

5.4 Implementation Considerations for Systolic ANNs

5.4.1 PE Designs for Retrieving and Learning Phases

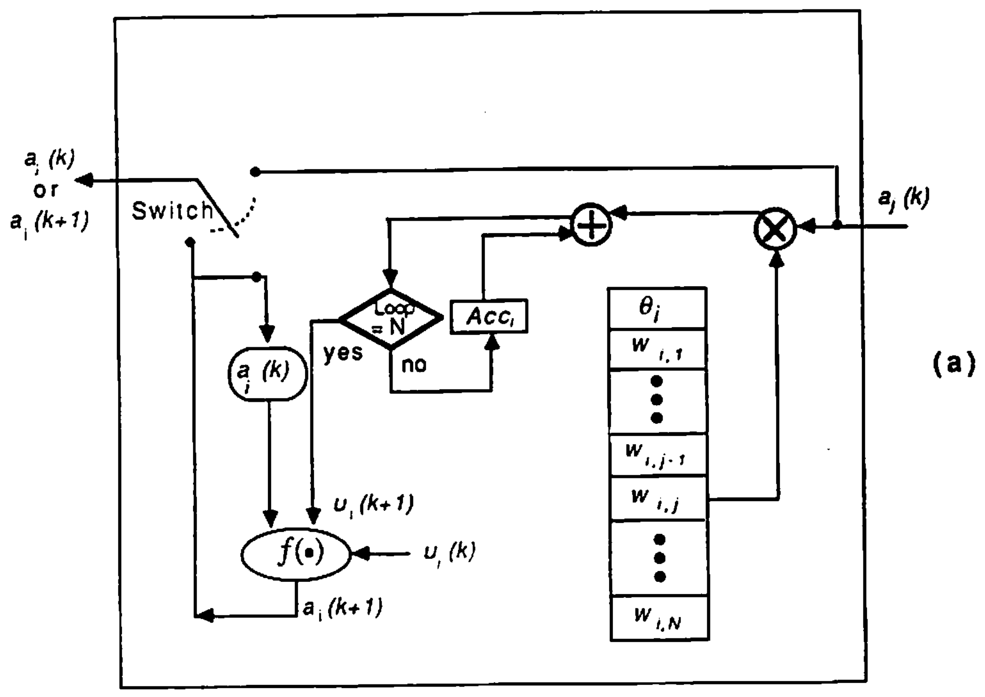
The schematic diagrams for the logical level design of a PE of the systolic ANN is shown in Figure 5.8. The synaptic weights $(w_{i1}, w_{i2}, \ldots, w_{iN})$ in the i-th PE are stored in RAM with simple circular-shift-up addressing. In the operations of the retrieving phase (see Eq. 1.3), the activation value $a_j(k)$ is produced at j-th PE, circularly propagated leftward in the ring systolic ANN, and multiplied by w_{ij} when it arrives at i-th PE. The product is then added into the accumulator Acc_i to form the net input $u_i(k+1)$. For the learning phase (see Eq. 1.5), almost the same hardware is adopted, but with somewhat different control sequences. Figure 5.8(b) shows the operations for a delta learning rule, where w_{ij} is retrieved from the RAM, modified by Δw_{ij} as computed, and then stored back to the RAM. All these operations are performed in one system clock.

5.4.2 Using INMOS T800 Transputer as a Building Block

There are two approaches to the design of a special purpose computing system. One is to design a programmable system. the other is a dedicated hard-wired system. The former one can be easily implemented by using commercial available VLSI processors chips.

The INMOS T800 Transputer is an Occam-based RISC architecture which provides 64-bit floating point arithmetic capability operating at 1 million multiply/add operations per second. The Transputer also has 4 Kbyte of on-chip RAM and can address up to 32 *Gbytes* of external memory [111,147]. In addition, the T800 Transputer provides four high-speed byte-serial hardwired bidirectional links operating at a 20 *Mbit/s* rate. Another reason for choosing a Transputer as the processing element is the Occam language. The Transputer is optimized to run Occam efficiently. Occam provides an easy way to translate our design into a Transputer program. The Transputer also efficiently supports other languages. such as C. Pascal. and Fortran. These features make the T800 a potential candidate for the building blocks of for our systolic ANNs.

For a more dedicated design using VLSI technology, there are two design alternatives that can be considered. In the following, we will only consider the fixed point operation option which is still the main focus of VLSI technology.



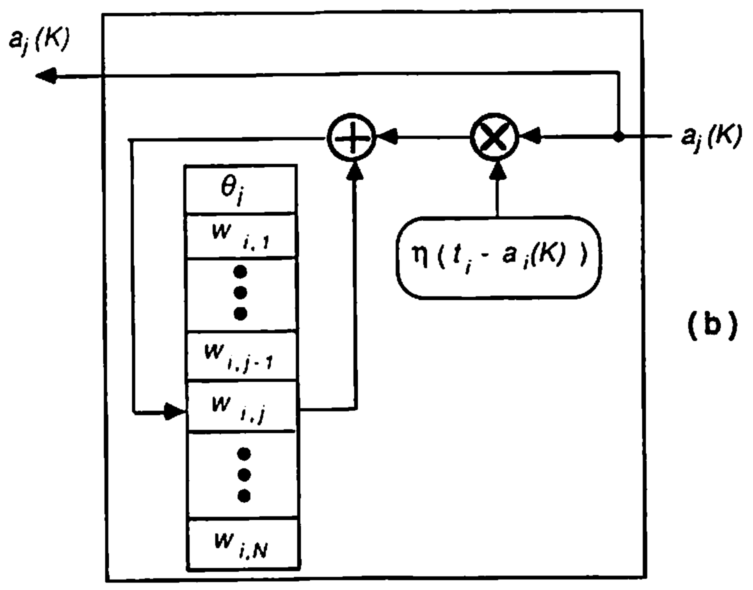


Figure 5.8: Schematic diagrams for the logical level PE designs in a systolic ANN. (a) Logical PE design for the retrieving phase. (b) Logical PE design for the learning phase.

5.4.3 Neural Processing Units Based on a Parallel Array Multiplier

Most of the computations involved in the neural processing units are multiply-and-accumulation (MAC) operations, so for a computationly efficient dedicated design, a parallel array multiplier should be used, e.g., Baugh-Wooley multiplier [2,16].

There are various forms of the nonlinear activation function adopted in neural nets, e.g., step function, squashing function, or sigmoid function. A simple comparator can be used to implement the step function and the array multiplier can be easily used to implement the squashing function. For digital implementation of the sigmoid function, fast table-look-up techniques should be adopted. In fact, a large portion of the sigmoid function is in linear range, so the look-up table is only used in the two extreme ranges of the sigmoid function.

5.4.4 Neural Processing Units Based on the CORDIC Processors

One disadvantage of using parallel array multipliers for neural processing units is that the area consumption might be significant. It has been found that for manipulating the same number of bits, a CORDIC (coordinate rotation digital computer) requires only about one-third of the silicon area of an array multiplier. A CORDIC provides almost the same execution speed as an array multiplier in performing vector-rotation operations, but much reduced execution speed in performing multiplication [2]. So for a area efficient dedicated design, a CORDIC processor might be more suitable. If the speed is more important than the area, bit-level pipelined CORDIC designs can also be considered [45,164].

A CORDIC scheme, as shown in Figure 5.9, is an iterative method based on bit-level shift-and-add operations. It is especially suitable for computing a specific set of functions including rotations of two dimensional vectors, trigonometric functions, logarithms, multiplications and divisions [166,33,2,81]. For example a 2-dimensional vector $\mathbf{v} = [x, y]$ can be rotated by an angle α by using a rotation operator R_m^α, i.e.,

$$\mathbf{v}' = R_m^\alpha \mathbf{v}$$

where

$$\begin{aligned} R_m^\alpha &= \begin{bmatrix} \cos\sqrt{m}\alpha & -\sqrt{m}\sin\sqrt{m}\alpha \\ \frac{1}{\sqrt{m}}\sin\sqrt{m}\alpha & \cos\sqrt{m}\alpha \end{bmatrix} \\ &= K \begin{bmatrix} 1 & -\sqrt{m}\tan\sqrt{m}\alpha \\ \frac{1}{\sqrt{m}}\tan\sqrt{m}\alpha & 1 \end{bmatrix} \end{aligned}$$

and the scaling constant $K = \cos\sqrt{m}\alpha$. The parameter m characterizes three possible arithmetic operations, namely the circular ($m = 1$), the linear ($m = 0$) and the hyperbolic ($m = -1$) operations [166,33,2,81].

In the linear coordinate system ($m = 0$), the CORDIC can be used for MAC operations (see Figures 5.10(a)), i.e., the rotation operator can be simplified as

$$R_0^{\alpha} = \begin{bmatrix} 1 & 0 \\ \alpha & 1 \end{bmatrix} \tag{5.7}$$

Given x, y, and α, we can get the CORDIC output $y + x\alpha$, which is very useful for propagation rule operations in the neural processing units.

In the hyperbolic coordinate system ($m = -1$), the rotation operator can be simplified to (see Figures 5.10(b)):

$$R_{-1}^{\alpha} = K \begin{bmatrix} 1 & \tanh\alpha \\ \tanh\alpha & 1 \end{bmatrix} \tag{5.8}$$

Note that the nonlinear sigmoid activation function bears the form:

$$f(u) = \frac{1}{1 + e^{-u}} = (1/2)(1 + \tanh(u/2)) \tag{5.9}$$

where u is the net input of the neuron. By comparing Eq. 5.8 and Eq. 5.9, it is easily seen that the sigmoid activation function can be implemented by setting $m = -1$, $\mathbf{v} = [1, 1]$ and $\alpha = u/2$. (The scaling operation K is not necessary here.)

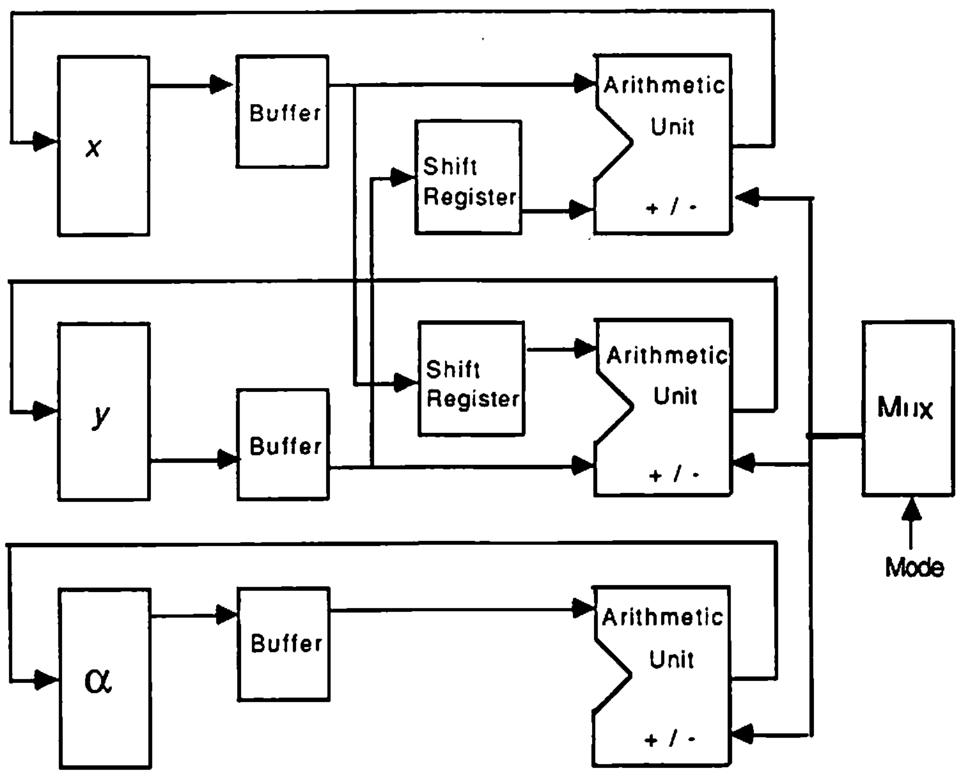


Figure 5.9: The arithmetic unit of a CORDIC processor.

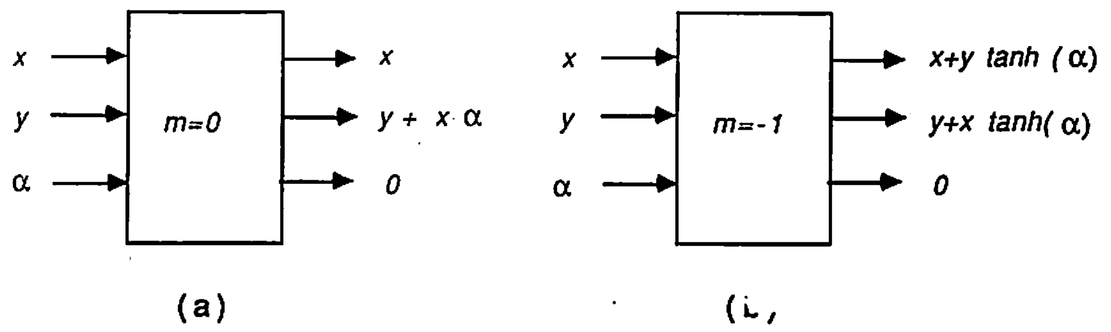


Figure 5.10: (a) The linear coordinate ($m = 0$) system of the CORDIC processing. (b) The hyperbolic coordinate ($m = -1$) system of the CORDIC processing.

Chapter 6

From Multilayer Perceptrons to Hidden Markov Models

This chapter extends the discussion of algorithmic and architectural studies of multilayer Perceptrons to hidden Markov models (HMMs). Our discussion starts with a unifying viewpoint of the retrieving (scoring) and learning phases of multilayer Perceptrons and HMMs. A ring systolic architecture is then proposed for implementing both the scoring and learning phases of HMMs.

6.1 A Unifying Viewpoint for the Multilayer Perceptrons and HMMs

From a retrieving phase point of view, hidden Markov models described by a trellis structure can be regarded as a special configurations of multilayer Perceptrons with a squashing type of nonlinear activation function. From a learning phase point of view, iterative gradient-descent-like approaches, which were used to derive BP learning in the multilayer Perceptrons, can successfully be applied to the trellis structure to derive the Baum-Welch reestimation formulation in HMMs.

6.1.1 A Special Configuration of Multilayer Perceptrons

Retrieving Phase

The system dynamics in the retrieving phase of a T-layer Perceptron with an equal number of neuron units (N) at each layer can be described by the following equations (see Figure 6.1):

$$\begin{aligned} u_i(t+1) &= \sum_{j=1}^{N} w_{ij}(t+1)a_j(t) + \theta_i(t+1) = \sum_{j=1}^{N+1} w_{ij}(t+1)a_j(t) & (6.1) \\ a_i(t+1) &= f(u_i(t+1)) \quad 1 \le i \le N, \quad 0 \le t \le T-1 & (6.2) \end{aligned}$$

Again for simplicity, the external input $\theta_i(t+1)$ is treated as a synaptic weight $w_{i,N+1}(t+1)$, which has a clamped input $a_{N+1}(t) = 1$.

Learning Phase

The learning phase of this T-layer Perceptron adopts the back propagation learning rule. The synaptic weights between the (t-th and ($t-1$)-th) layers can be updated recursively (in the order of $t = T,\ T-1,\ \ldots,\ 1$) :

$$w_{ij}^{(m+1)}(t) = w_{ij}^{(m)}(t) + \eta\ \delta_i^{(m)}(t)\ a_j^{(m)}(t-1) \qquad (6.3)$$

where the error signals $\delta_i^{(m)}(t)$ are recursively computed as follows (in the order of $t = T, T-1, \ldots, 1$):

$$\text{if } t = T, \quad \delta_i^{(m)}(T) = f'(u_i^{(m)}(T))\,(t_i^{(m)} - a_i^{(m)}(T)) \tag{6.4}$$

$$\text{if } t < T, \quad \delta_i^{(m)}(t) = f'(u_i^{(m)}(t)) \sum_{j=1}^{N} \delta_j^{(m)}(t+1)\, w_{ji}^{(m)}(t+1) \tag{6.5}$$

6.1.2 The Hidden Markov Model

A hidden Markov model (HMM) is a doubly stochastic process with an underlying stochastic process that is not observable (i.e., hidden), but can only be observed through another set of stochastic processes that produce the sequence of observed symbols [134]. HMM techniques have been successfully applied to speech recognition tasks [13,67,62,134]. The technique is also similar to the ***dynamic time warping*** approaches of speech recognition, which uses temporal alignment techniques and addresses the nonlinearity in the speech signals [67]. There are other potential applications of HMMs. e.g., for English spelling checking [55], and for hierarchical character recognition [165]. The basic components of an HMM can be briefly described as follows:

1. There are N possible states $\{q_i,\ i = 1,\ 2,\ \ldots,\ N\}$ in an HMM. The new state is entered at time t based upon a ***state transition probability*** a_{ij}, which depends on the previous state at time $t-1$ (the Markovian property), i.e.,

$$a_{ij} = Pr(\ q_i \text{ at } t \mid q_j \text{ at } t-1)$$

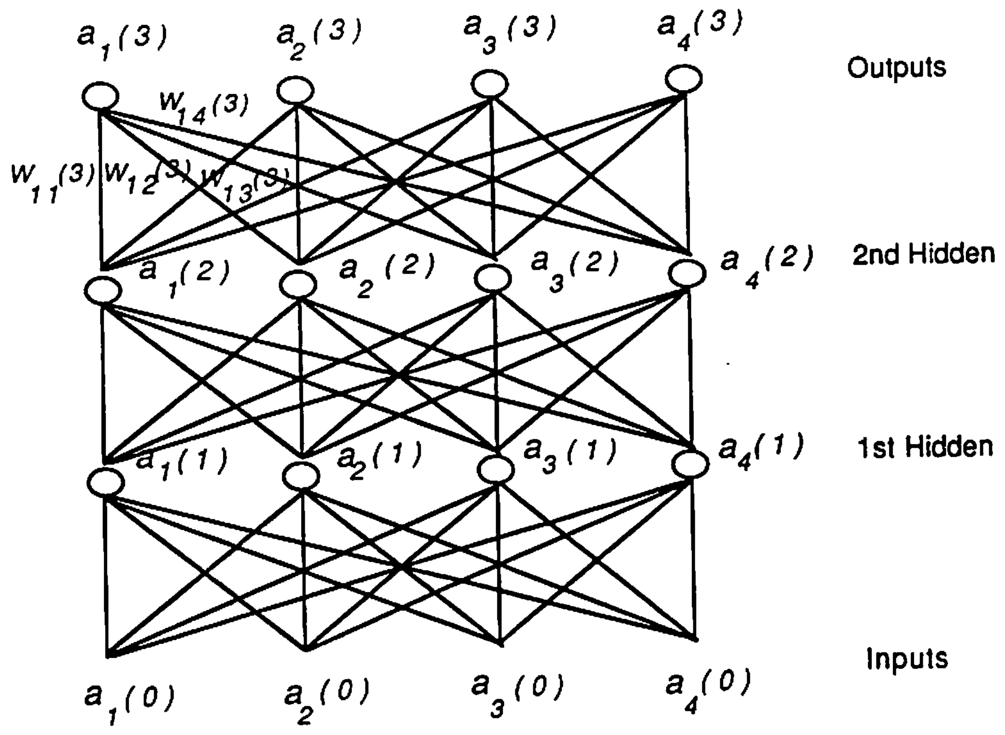


Figure 6.1: A three-layer Perceptron consists of the two hidden layers and the output layer. Each layer has 4 neuron units.

2. The state transition probability at time $t = 0$ is termed *initial state probability* π_i, i.e..

$$\pi_i = Pr(\ q_i \text{ at } t = 0)$$

3. Given the state transition probabilities, there are M possible symbols $\{v_i,\ i = 1, 2, \ldots, M\}$ that can be observed. and a *symbol occurrence probability* is determined according to a probability distribution which depends on the current state. Each symbol, say k-th symbol, at state i has its symbol occurrence probability $b_i(k)$:

$$b_i(k) = Pr(\ v_k \text{ at } t \mid q_i \text{ at } t)$$

4. Given an HMM, where $\{a_{ij}\}$, $\{b_i(k)\}$ and $\{\pi_i\}$ are specified, $\lambda = (A, B, \pi)$, an observation sequence O with length $T+1$ is generated, $O = (o_0, o_1, \cdots o_T)$.

Scoring and Learning Phases of an HMM

The algorithms involved in an HMM can be divided into two phases: *scoring* and *learning* [58,59]. Given an observation sequence $O = (o_0, o_1, \cdots o_T)$, and a pre-specified HMM $\lambda = (A, B, \pi)$, the most important computation involved in the scoring phase of an HMM is to compute the scoring probability $Pr(O|\lambda)$, which allows us to choose one among several models that best matches the observations. The computation involved in the learning phase of an HMM is to adjust the model parameters (A, B, π), so that we can maximize $Pr(O|\lambda)$ given the training sequence O. The learning phase allows us to optimally adapt model parameters to the observed training data. i.e., to create the best model for real phenomena.

The Forward Evaluation Procedure In order to calculate the scoring probability $Pr(O|\lambda)$ of the observation sequence O, given the model λ, a very efficient procedure called the *forward evaluation procedure* has been widely used [134]. Let us define the forward likelihood $\alpha_t(i)$ to be the probability of the partial observation sequence (until time t) and state q_i at time t, given the model λ, i.e.,

$$\alpha_t(i) = Pr(o_0, o_1, \ldots o_t, i_t = q_i \mid \lambda)$$

where $\alpha_t(i)$ can be calculated inductively [134]:

$$\alpha_0(i) = \pi_i \, b_i(o_0), \quad 1 \leq i \leq N \tag{6.6}$$

$$\alpha_{t+1}(i) = \left[\sum_{j=1}^{N} a_{ij} \, \alpha_t(j)\right] b_i(o_{t+1}), \quad 0 \leq t \leq T-1, \quad 1 \leq i \leq N \tag{6.7}$$

$$Pr(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{6.8}$$

The Backward Evaluation Procedure In a similar manner, the backward likelihood $\beta_t(i)$, which is useful in the computations of the learning phase, is defined as the probability of the partial observation sequence from $t+1$ to the end, given state q_i at time t and the model λ, i.e.,

$$\beta_t(i) = Pr(o_{t+1}, o_{t+2}, \ldots o_T \mid i_t = q_i, \lambda)$$

again $\beta_t(i)$ can be calculated inductively:

$$\beta_T(i) = 1, \quad 1 \leq i \leq N \tag{6.9}$$

$$\beta_t(i) = \sum_{j=1}^{N} \beta_{t+1}(j)\, b_j(o_{t+1})\, a_{ji}, \quad T-1 \geq t \geq 0, \quad 1 \leq i \leq N \tag{6.10}$$

Trellis Structure for the Evaluation Procedures The forward-backward evaluation technique takes advantage of a trellis structure to reduce the computational burden. The computational complexity of the evaluation of the scoring probability $Pr(O|\lambda)$ can be illustrated by a tree structure as shown in Figure 6.2(a), which shows the exponential growth of complexity in T. This tree structure treats distinctive paths differently, as if at instance t, the number of available state indices were N^t, as denoted by the parenthesized index in Figure 6.2(a). Instead of traversing all the nodes in the tree structure, which requires $O(2T \cdot N^T)$ calculations to obtain the scoring probability $Pr(O|\lambda)$, the forward-backward evaluation procedures transform the tree structure in Figure 6.2(a) into a trellis structure shown in Figure 6.2(b), by recognizing the fact that at any instance t and any state q_i there are always only N possible next states regardless of the past transition history. The branches in the tree structure merge into N nodes (states) at every instance [66], and the computation is reduced to $O(N^2T)$ calculations [134].

Viterbi Algorithm in the Scoring Phase By transforming the tree structure into the trellis structure, dynamic programming techniques, e.g., the Viterbi

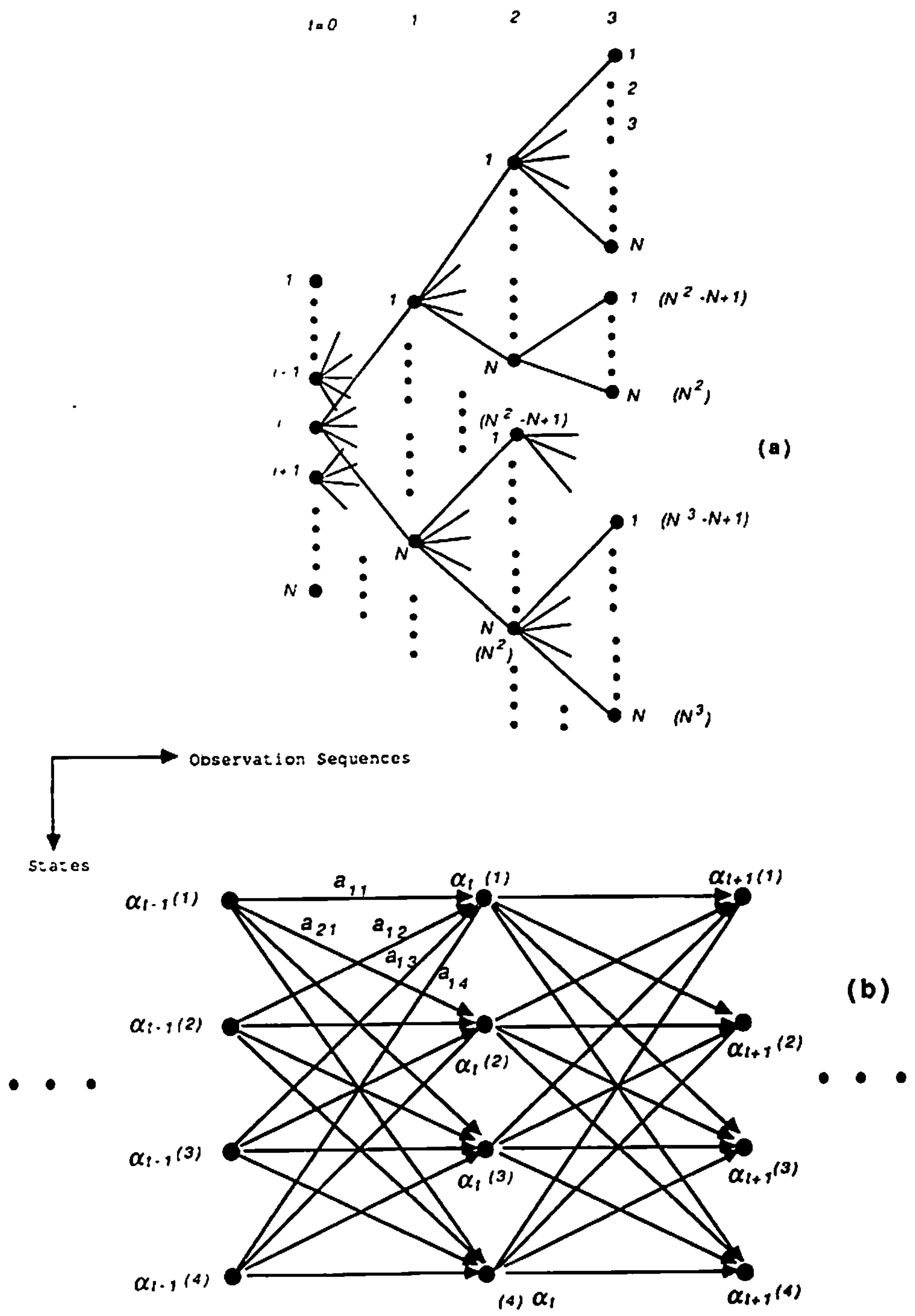


Figure 6.2: (a) Tree structure for evaluation of the scoring probability $Pr(O|\lambda)$. (b) Trellis structure for evaluating the scoring probability in an HMM.

algorithm, can now be efficiently employed to find the best state sequence $I^* = (i_0^*, i_1^*, \ldots i_T^*)$ such that:

$$Pr(O, I^*|\lambda) = \max_{\text{all } I} Pr(O, I|\lambda)$$

The formal steps in the Viterbi algorithm for finding the single best state sequence can be described below [134]:

1. **Initialization:**

$$\begin{aligned} \text{for} \quad & 1 \leq i \leq N \\ \delta_0(i) &= \pi_i\, b_i(o_0) \\ \psi_0(i) &= 0 \end{aligned} \tag{6.11}$$

2. **Recursion:**

$$\begin{aligned} \text{for} \quad & 0 \leq t \leq T-1, \quad 1 \leq j \leq N \\ \delta_{t+1}(i) &= \max_{1 \leq j \leq N} [a_{ij}\delta_t(j)] b_i(o_{t+1}) \\ \psi_{t+1}(i) &= \arg\max_{1 \leq j \leq N} [a_{ij}\delta_t(j)] \end{aligned} \tag{6.12}$$

3. **Termination:**

$$\begin{aligned} p^* &= \max_{1 \leq i \leq N} [\delta_T(i)] \\ i_T^* &= \arg\max_{1 \leq i \leq N} [\delta_T(i)] \end{aligned} \tag{6.13}$$

4. **Path Backtracking:**

$$\begin{aligned}
\text{for } t &= T-1,\ T-2,\ \ldots,\ 0 \\
i_t^* &= \varphi_{t+1}(i_{t+1}^*) \qquad (6.14)
\end{aligned}$$

Note that the Viterbi algorithm is similar (without the path backtracking steps) in implementation to the forward evaluation procedure. However, a maximization over previous states is used in place of the summing operation used in the evaluation procedure. Again the trellis structure can be used to efficiently implement the computation [134].

Reestimation Algorithm in the Learning Phase

The operations involved in the learning phase of an HMM adj[illegible] the model parameters (A, B, π) to maximize the scoring probability of the observation sequence given the model. A widely used iterative procedure is the ***Baum-Welch reestimation*** algorithm [20,19]. In using this reestimation algorithm, two probability working variables needed to be introduced;

$$\begin{aligned}
\gamma_t(i) &= Pr(i_t = q_i, O|\lambda) \\
&= \alpha_t(i)\ \beta_t(i) \qquad (6.15) \\
\xi_t(i,j) &= Pr(i_t = q_i, i_{t-1} = q_j, O|\lambda) \\
&= a_{ij}\ \alpha_{t-1}(j)\ b_i(o_t)\ \beta_t(i) \qquad (6.16)
\end{aligned}$$

where $\gamma_t(i)$ is the probability of being in state q_i and in presence of the training sequence O at time t, given the model λ. $\xi_t(i,j)$ is the probability of a path being in state q_j also in presence of the training sequence O at time $t-1$ and making a transition to state q_i at time t, given the model λ. The reestimation formula for A, B and π can thus be formulated:

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T} \xi_t(i,j)}{\sum_{t=0}^{T-1} \gamma_t(j)} \tag{6.17}$$

$$\bar{b}_i(k) = \frac{\sum_{t=0,\ o_t=k}^{T} \gamma_t(i)}{\sum_{t=0}^{T} \gamma_t(i)} \tag{6.18}$$

$$\bar{\pi}_i = \gamma_1(i) \tag{6.19}$$

If we define the initial model as $\lambda = (A,\ B,\ \pi)$, and the reestimated model as $\bar{\lambda} = (\bar{A},\ \bar{B},\ \bar{\pi})$, then it can be proven that from each parameter reestimation, we have found a new model $\bar{\lambda}$ from which the training sequence is more likely to be produced [134], i.e.,

$$Pr(O|\bar{\lambda}) > Pr(O|\lambda).$$

The reestimation algorithm can then be applied to the given training sequence iteratively so that we can successively acquire new $\{\bar{\lambda}\}$, which improves the scoring probability of O being observed from the model until some convergent point is reached.

6.1.3 Relationship Between a Multilayer Perceptron and an HMM

HMMs, which use the Baum-Welch reestimation learning algorithm (a maximum likelihood estimator [66]), have been widely used in speech recognition tasks. One of the main advantages of using HMMs for speech recognition lies in the possibility to take account of the time sequential order of speech signals and to include the time warping process. However, the *a priori* choice of the model topology (e.g., number of states, allowable transitions, probability distribution and transition rules) limits the flexibility of the models and make it difficult to include non-explicit knowledge of speech production and recognition processes [23].

On the other hand, the back-propagation learning algorithm used in the multilayer Perceptrons performs an iterative gradient descent search based on minimizing the least squares error function in the synaptic weight space. When compared with the HMMs, the main distinction of this algorithm is its ability to memorize and generalize high-order constraints (implicit knowledge) between data while stressing the discrimination power between the different classes [23]. For example, BP learning compares favorably with Baum-Welch reestimation learning at the task of distinguishing the word "bee," "dee," "ee," and "vee" spoken by many different male speakers in a noisy environment [168].

In the following, the T layer Perceptron specified by Eq. 6.1 and Eq. 6.2 is used for demonstrating the equivalence of Perceptrons and HMMs. The comparison will be made for both the retrieving (scoring) and learning phases of the two models [57].

Relationship in the Scoring (Retrieving) Phase

The most important algorithm in the scoring phase of an HMM is the forward evaluation computation. see Section 6.1.2. Note that the Viterbi algorithm has almost the same computational structure as the forward evaluation procedure, so the following discussion will be mainly focused on the forward evaluation algorithm. In the following. we will show that by equating some variables, and defining a special nonlinear activation function. the forward computation in the retrieving phase of a multilayer Perceptron is indeed executing the forward evaluation (or the Viterbi algorithm) of an HMM.

Retrieving Phase of a Multilayer Perceptron The system dynamics in the retrieving phase of the above multilayer Perceptron can be rewritten in an iterative formulation,

$$\begin{aligned} u_i(t+1) &= \sum_{j=1}^{N} w_{ij}(t+1)\, a_j(t) + \theta_i(t+1) \\ &= \sum_{j=1}^{N+1} w_{ij}(t+1)\, a_j(t) \\ a_i(t+1) &= f(u_i(t+1)) \quad t = 0,\ 1,\ \cdots,\ T-1 \end{aligned} \tag{6.20}$$

where the external input $\theta_i(t+1)$ can be thought to control the operating point of the nonlinear activation function.

Scoring Phase of an HMM As discussed in Section 6.1.2. the forward evaluation computation in an HMM can also be expressed in an iterative formulation,

$$\begin{aligned} u_{t+1}(i) &= \sum_{j=1}^{N} a_{ij}\ \alpha_t(j) \\ \alpha_{t+1}(i) &= u_{t+1}(i)\ b_i(o_{t+1}) \quad t = 0,\ 1,\ \cdots,\ T-1 \end{aligned} \tag{6.21}$$

where the inputs are $\alpha_0(i) = \pi_i\ b_i(o_0)$. and the nonlinear operation (discrete or continuous) is performed by the $b_i(\cdot)$ mapping function giving the external inputs (observation symbols) $\{o_{t+1}\}$. Also. it is worthy of mention that the same set of transition probabilities $\{a_{ij}\}$ is used for all of the layers.

Based on Eq. 6.20 and Eq. 6.21. it can be easily seen that the forward evaluation in the scoring phase of an HMM can be regarded as a special case of the retrievi. ͜ phase of a multilayer Perceptron with a new definition of a squashing-like nonlinear activation function.

Relationship in the Learning Phase

There are very high correlations that can be detected between the mathematical formulations of BP learning for multilayer Perceptrons and Baum-Welch reestimation for HMMs [22]. In the following, we will show that by proper variable equating, the iterative gradient-descent-like approach can be used to derive the mathematical formulation of the Baum-Welch reestimation algorithm, which was originally derived from statistical estimation techniques [20,19,108,66].

Derivations of BP Learning The mathematical derivations of BP learning algorithm for a T-layer Perceptron basically follows an iterative gradient descent approach. Given the m-th pair of training patterns, $\{\mathbf{a}^{(m)}(0), \mathbf{t}^{(m)}, m = 1, 2, \cdots, M\}$, our goal is to iteratively choose a set of $\{w_{ij}(t), t = 1, 2, \ldots, T\}$ for all layers so that the cost (error) function ζ can be minimized. More specifically,

$$\begin{aligned} w_{ij}^{(m+1)}(t) &= w_{ij}^{(m)}(t) + \Delta w_{ij}^{(m)}(t) && (6.22) \\ &= w_{ij}^{(m)}(t) - \eta \frac{\partial \zeta}{\partial w_{ij}^{(m)}(t)} && (6.23) \end{aligned}$$

where

$$\zeta = \frac{1}{2} \sum_{m=1}^{M} \sum_{i=1}^{N} (t_i^{(m)} - a_i^{(m)}(T))^2$$

By using the chain rule,

$$\begin{aligned} -\frac{\partial \zeta}{\partial w_{ij}^{(m)}(t)} &= -\frac{\partial \zeta}{\partial u_i^{(m)}(t)} \frac{\partial u_i^{(m)}(t)}{\partial w_{ij}^{(m)}(t)} \\ &= \delta_i^{(m)}(t)\, a_j^{(m)}(t-1) && (6.24) \end{aligned}$$

where the back-propagated error signal $\delta_i^{(m)}(t)$ is defined as

$$\begin{aligned} \delta_i^{(m)}(t) &= -\frac{\partial \zeta}{\partial u_i^{(m)}(t)} \\ &= -\frac{\partial \zeta}{\partial a_i^{(m)}(t)} \frac{\partial a_i^{(m)}(t)}{\partial u_i^{(m)}(t)} \\ &= -\frac{\partial \zeta}{\partial a_i^{(m)}(t)} f'(u_i^{(m)}(t)) \end{aligned}$$

$$
\begin{aligned}
&= -[\sum_{j=1}^{N} \frac{\partial \zeta}{\partial u_j^{(m)}(t+1)} \frac{\partial u_j^{(m)}(t+1)}{\partial a_i^{(m)}(t)}] f'(u_i^{(m)}(t)) \\
&= [\sum_{j=1}^{N} \delta_j^{(m)}(t+1)\, w_{ji}^{(m)}(t+1)]\, f'(u_i^{(m)}(t)) \qquad (6.25)
\end{aligned}
$$

Note that the error signal on the top layer $\delta_i^{(m)}(T)$ can be computed directly without recursive formulation.

$$
\begin{aligned}
\delta_i^{(m)}(T) &= -\frac{\partial \zeta}{\partial u_i^{(m)}(T)} \\
&= -\frac{\partial \zeta}{\partial a_i^{(m)}(T)} \frac{\partial a_i^{(m)}(T)}{\partial u_i^{(m)}(T)} \\
&= (t_i^{(m)} - a_i^{(m)}(T))\, f'(u_i^{(m)}(T)) \qquad (6.26)
\end{aligned}
$$

Based on Eq. 6.23, Eq. 6.26, Eq. 6.24, and Eq. 6.25, we can get the updating formulation for the synaptic weights (also external inputs):

$$
\begin{aligned}
w_{ij}^{(m+1)}(T) &= w_{ij}^{(m)}(T) + \eta\, f'(u_i^{(m)}(T))\, (t_i^{(m)} - a_i^{(m)}(T))\, a_j^{(m)}(T-1) \\
w_{ij}^{(m+1)}(t) &= w_{ij}^{(m)}(t) + \eta\, f'(u_i^{(m)}(t))\, [\sum_{j=1}^{N} \delta_j^{(m)}(t+1)\, w_{ji}^{(m)}(t+1)]\, a_j^{(m)}(t-1)
\end{aligned}
$$

Derivation of Baum-Welch Reestimation Learning In the following, we will show that the iterative gradient descent approach used to derive the formulation of BP learning in a multilayer Perceptron can also be adapted to derive the Baum-Welch reestimation learning used in an HMM [57]. Given the observation sequence O, our goal is to iteratively choose a set of $\{a_{ij}^{(m)}\}$ so that the likelihood $\hat{\zeta}$ can be maximized. Due to the probabilistic nature of an HMM, the incremental additive updating of the synaptic weights in the BP learning is replaced by the incremental multiplicative updating of the transition probability:

$$
\begin{aligned}
a_{ij}^{(m+1)} &= a_{ij}^{(m)} \cdot \Delta a_{ij}^{(m)} \\
&= \eta' \, a_{ij}^{(m)} \frac{\partial \hat{\zeta}}{\partial a_{ij}^{(m)}}
\end{aligned}
$$

where

$$
\hat{\zeta} = Pr(O|\lambda) = \sum_{j=1}^{N} \alpha_T^{(m)}(j)
$$

Again by using the chain rule.

$$
\begin{aligned}
\frac{\partial \hat{\zeta}}{\partial a_{ij}^{(m)}} &= \sum_{t=1}^{T} \frac{\partial \hat{\zeta}}{\partial \alpha_t^{(m)}(i)} \frac{\partial \alpha_t^{(m)}(i)}{\partial a_{ij}^{(m)}} \\
&= \sum_{t=1}^{T} \beta_t^{(m)}(i) \frac{\partial \alpha_t^{(m)}(i)}{\partial a_{ij}^{(m)}} \\
&= \sum_{t=1}^{T} \beta_t^{(m)}(i) \frac{\partial \alpha_t^{(m)}(i)}{\partial u_i^{(m)}(t)} \frac{\partial u_i^{(m)}(t)}{\partial a_{ij}^{(m)}} \\
&= \sum_{t=1}^{T} \beta_t^{(m)}(i) \, b_i^{(m)}(o_t) \, \alpha_{t-1}^{(m)}(j)
\end{aligned}
$$

By combining Eq. 6.27 and 6.27. the iterative updating formulation for $a_{ij}^{(m)}$ can be obtained,

$$
\begin{aligned}
a_{ij}^{(m+1)} &= \eta' \, a_{ij}^{(m)} \frac{\partial \hat{\zeta}}{\partial a_{ij}^{(m)}} \\
&= \eta' \sum_{t=1}^{T} a_{ij}^{(m)} \beta_t^{(m)}(i) \, b_i^{(m)}(o_t) \, \alpha_{t-1}^{(m)}(j) \\
&= \eta' \sum_{t=1}^{T} \xi_t^{(m+1)}(i,j)
\end{aligned}
$$

where $\xi_t^{(m+1)}(i,j) = Pr(i_t = q_i, i_{t-1} = q_j, O|\lambda)$. Note that since only one set of weights used for all the T layers of trellis structure, so average of the intermediate weights $\xi_t^{(m+1)}(i,j)$ over all the layers is taken place. The backward likelihood signal $\beta_t^{(m)}(i)$, which is similar to the back propagated error signal in BP learning, is also introduced.

$$\begin{aligned}
\beta_t^{(m)}(i) &= \frac{\partial \hat{\zeta}}{\partial \alpha_t^{(m)}(i)} \\
&= \sum_{j=1}^{N} \frac{\partial \hat{\zeta}}{\partial u_{t+1}^{(m)}(j)} \frac{\partial u_{t+1}^{(m)}(j)}{\partial \alpha_t^{(m)}(i)} \\
&= \sum_{j=1}^{N} \frac{\partial \hat{\zeta}}{\partial \alpha_{t+1}^{(m)}(j)} \frac{\partial \alpha_{t+1}^{(m)}(j)}{\partial u_{t+1}^{(m)}(j)} a_{ji}^{(m)} \\
&= \sum_{j=1}^{N} \beta_{t+1}^{(m)}(j)\, b_j^{(m)}(o_{t+1})\, a_{ji}^{(m)} \qquad (6.27)
\end{aligned}$$

Note that the backward likelihood signal on the top layer $\beta_T^{(m)}(i)$ can be computed directly without recursive formulation, i.e., $\beta_T^{(m)}(i) = 1$.

Unlike the BP learning where no specific rule can be followed to select the learning rate, the updating step η' in the Baum-Welch reestimation algorithm is constrained by the standard Markovian property,

$$\begin{aligned}
\sum_{i=1}^{N} a_{ij}^{(m)} &= 1. \quad j = 1,\ 2,\ \cdots,\ N \\
&= \sum_{i=1}^{N} \eta' \sum_{t=1}^{T} \xi_t^{(m)}(i,j) \\
&= \eta' \sum_{i=1}^{N}\sum_{t=1}^{T} a_{ij}^{(m-1)}\, \beta_t^{(m)}(i)\, b_i^{(m)}(o_t)\, \alpha_{t-1}^{(m)}(j)
\end{aligned}$$

$$\begin{aligned}
&= \eta' \sum_{t=1}^{T} [\sum_{i=1}^{N} a_{ij}^{(m-1)} \beta_t^{(m)}(i)\, b_i^{(m)}(o_t)]\, \alpha_{t-1}^{(m)}(j) \\
&= \eta' \sum_{t=1}^{T} \beta_{t-1}^{(m)}(j)\, \alpha_{t-1}^{(m)}(j) \\
&= \eta' \sum_{t=1}^{T} \gamma_{t-1}^{(m)}(j) \\
&= \eta' \sum_{t=0}^{T-1} \gamma_t^{(m)}(j) \qquad (6.28)
\end{aligned}$$

where $\gamma_t^{(m)}(j) = Pr(i_t = q_i, O|\lambda)$, and

$$\eta' = \frac{1}{\sum_{t=0}^{T-1} \gamma_t^{(m)}(j)}$$

6.2 Systolic Architectures for HMMs

6.2.1 Array Design for the Scoring Phase of an HMM

Similar to the ring systolic design for ANNs, the recursive forward evaluation procedure for the scoring phase of an HMM as given in Eq. 6.21 can also be rewritten as a generalized consecutive matrix-vector multiplication [93,95]:

$$\vec{\alpha}_0 = \vec{\pi} \odot \mathbf{b}(o_0) \qquad (6.29)$$

$$\vec{\alpha}_{t+1} = [\mathbf{A}\vec{\alpha}_t] \odot \mathbf{b}(o_{t+1}) \qquad (6.30)$$

where the $\odot$ operator performs corresponding element multiplication between two vectors, i.e.,

$$[x_1,\ x_2,\ \ldots,\ x_N]^T \odot [y_1,\ y_2,\ \ldots,\ y_N]^T = [x_1 y_1,\ x_2 y_2,\ \ldots,\ x_N y_N]^T$$

The forward likelihood vector $\vec{\alpha}_t$, initial state vector $\vec{\pi}$, symbol occurrence vector $\mathrm{b}(o_t)$, and the state transition matrix $\mathbf{A}$, which are used in Eqs. 6.29 and 6.30, are given by

$$\begin{aligned}
\vec{\alpha}_t &= [\alpha_t(1),\ \alpha_t(2),\ \ldots,\ \alpha_t(N)]^T \\
\vec{\pi}_t &= [\pi_1,\ \pi_2,\ \ldots,\ \pi_N]^T \\
\mathrm{b}(o_t) &= [b_1(o_t),\ b_2(o_t),\ \ldots,\ b_N(o_t)]^T \\
\mathbf{A} &= \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{bmatrix}
\end{aligned}$$

Similar to the DG design for the systolic ANN, the cascaded DG for HMM is depicted in Figure 6.3(a). This again leads to a ring systolic architecture as shown in Figure 6.3(b). The pipelining period of this HMM systolic array is 1, which implies 100% utilization efficiency during the iteration process of the forward evaluation procedure [81,99].

Forward Evaluation Procedure on the HMM Systolic Array

In the implementations of the forward evaluation procedure using the HMM systolic array, each PE, say the i-th PE, can be treated as a *state*, and the corresponding state transition probability $(a_{i1},\ a_{i2},\ \ldots, a_{iN})$ are stored in the memory with a circular-shift-up ordering as shown in Figure 6.4. The symbol occurrence probabilities $\{b_i(k)\}$ are also stored in i-th PE. A table-look-up mechanism is

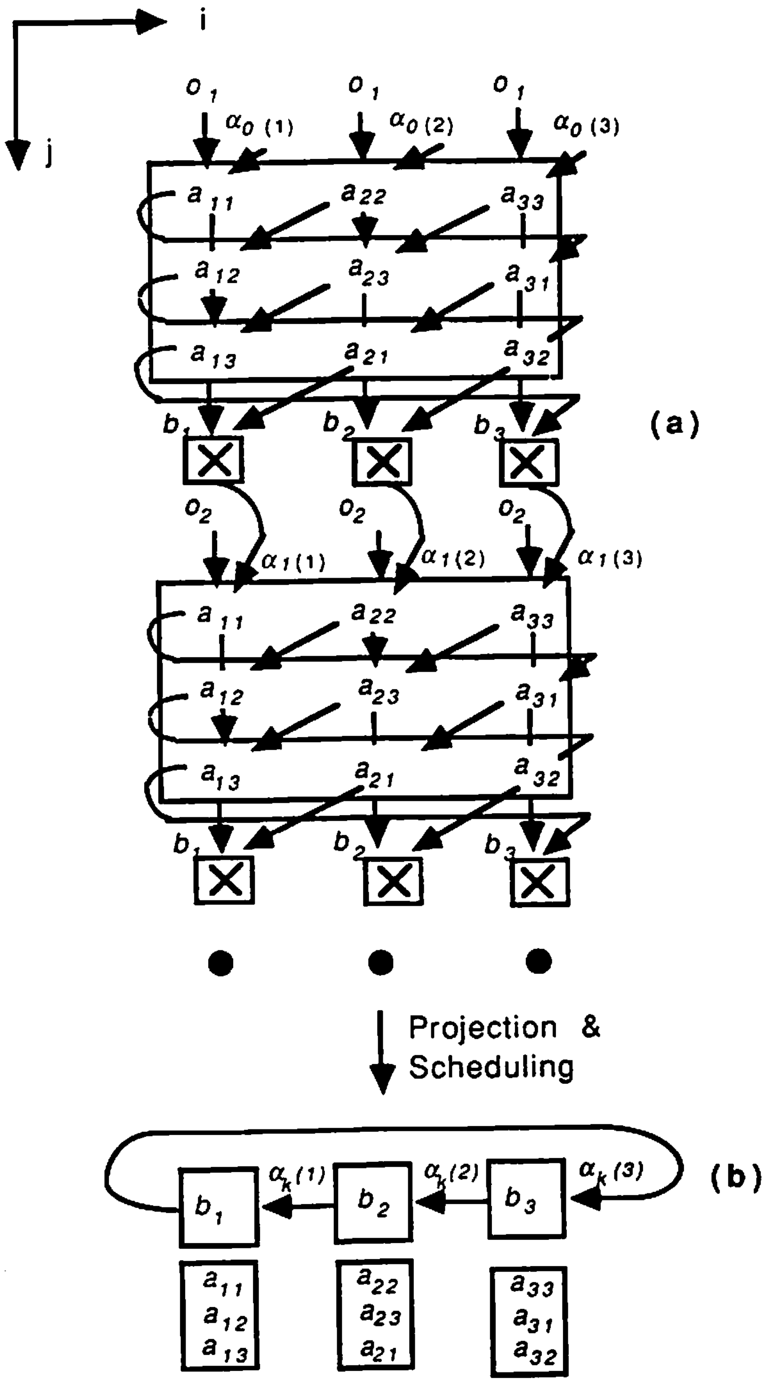


Figure 6.3: (a) Cascaded DG for the forward evaluation procedure in an HMM. (b) Ring systolic array for an HMM.

provided for fast associative retrieval of the symbol occurrence probability $b_i(o_t)$ when symbol o_t is observed at time instant t. The operations required of each PE in order to perform the forward evaluation procedure are described as follows (see Eqs. 6.6, 6.7, 6.8).

1. **Initialization:** Each PE, say the i-th PE, receives π_i and o_1 from its corresponding input channel to compute its initial forward likelihood $\alpha_0(i) = \pi_i \; b_i(o_0)$ in one clock unit. This initial value $\alpha_0(i)$ is then multiplied with a_{ii} to form the product which is assigned to the accumulating variable μ_i. This initiates the iteration cycle at time $t = 1$. The initial value $\alpha_0(i)$ is then pipelined leftward to be cycled in the HMM systolic array.

2. **Recursion:** Each PE, say the i-th PE, upon receiving $\alpha_t(j)$ at each clock of the t-th cycle will also fetch the corresponding transition probability a_{ij} from the cyclic-shift memory in synchrony, and will form the product to be combined with the μ_i stored in the i-th PE. After N clock units, the net input $u_{t+1}(i)$ is completely formed.

$$u_{t+1}(i) = \mu_i \text{ after N clocks } = \sum_{j=1}^{N} a_{ij}\alpha_t(j)$$

   The observation symbol o_{t+1} is then read in from the input port, after the complete accumulation of $u_{t+1}(i)$, to find the symbol observation probability $b_i(o_{t+1})$. This probability is immediately multiplied by $u_{t+1}(i)$ to yield the next forward likelihood $\alpha_{t+1}(i)$. This new forward likelihood is again multiplied by a_{ii} to yield a new μ_j, starting the next $(t+1)$-th cycle.

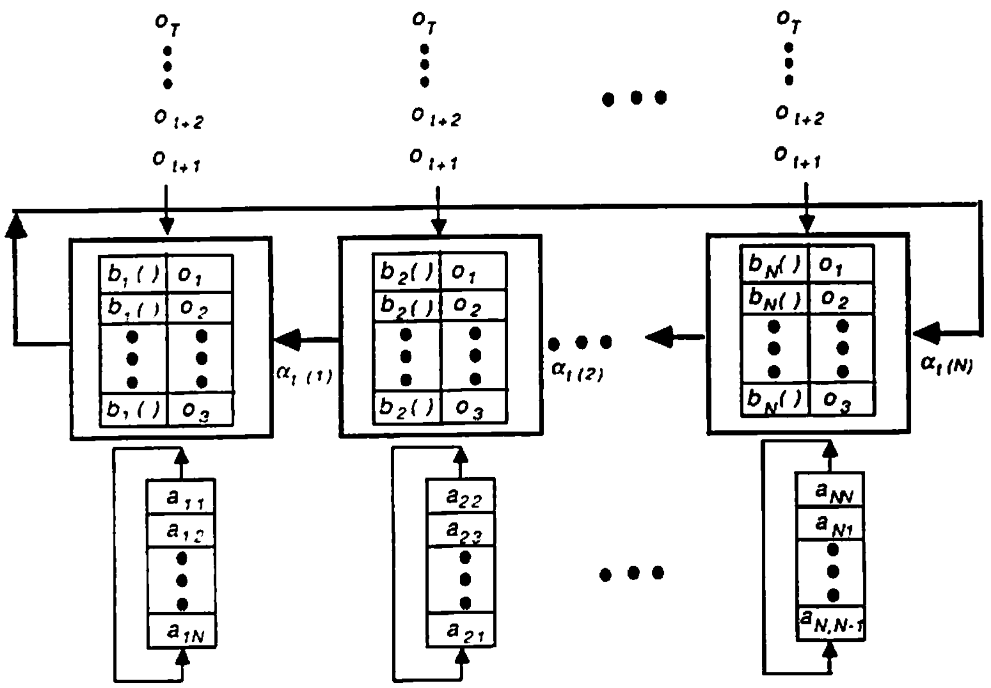


Figure 6.4: The execution of the forward evaluation procedure on the HMM systolic array.

3. **Termination:** After T cycles have passed, each $\alpha_T(i)$, created at the i-th PE, is pipelined leftward for one more cycles to generate the desired scoring probability $Pr(O|\lambda)$ at all the PEs, i.e.,

$$Pr(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i).$$

Backward Evaluation Procedure on the HMM Systolic Array

The same systolic array for the HMM forward evaluation procedure can be easily adapted to implement the backward evaluation procedure. The operations required of each PE in order to perform the backward evaluation procedure are described as follows (see Figure 6.5).

1. **Initialization:** Each PE, say the i-th PE, receives o_T from its corresponding input channel to compute the its initial probability $b_i(o_T)$ in one clock unit. This initial value $b_i(o_T)$ is multiplied by a_{ii} to form the product which is assigned to the accumulating variable ρ_i. This initiates the iteration cycle at time $T-1$ and ρ_i is then pipelined leftward to be cycled in the HMM systolic array.

2. **Recursion:** Each PE, say the j-th PE, upon acquiring the newly generated $\beta_{t+1}(j)$ and receiving o_{t+1} from the input channel, will multiply $\beta_{t+1}(j)$ by $b_j(o_{t+1})$ and assign the product to a temporal variable ω_j. Whenever an accumulator ρ_i (sent from i-th PE) arrives, the variable ω_j is multiplied by a_{ji} and the product is added into ρ_i.

   $$\rho_i = \rho_i + \omega_j \; a_{ji}$$

   After N clock units, the accumulator ρ_i will return to the i-th PE with the required partial summation, and will be assigned to $\beta_t(i)$,

   $$\beta_t(i) = \rho_i \text{ at } i\text{-th PE } = \sum_{j=1}^{N} \omega_j a_{ji} = \sum_{j=1}^{N} [\beta_{t+1}(j) b_j(o_{t+1})] a_{ji}$$

   which will start another new cycle.

Viterbi Algorithm on the HMM Systolic Array

As discussed in Section 6.1.2, the Viterbi algorithm can be efficiently employed to find the particular state sequence $I^* = (i_0^*,\ i_1^*,\ \ldots\ i_T^*)$ such that

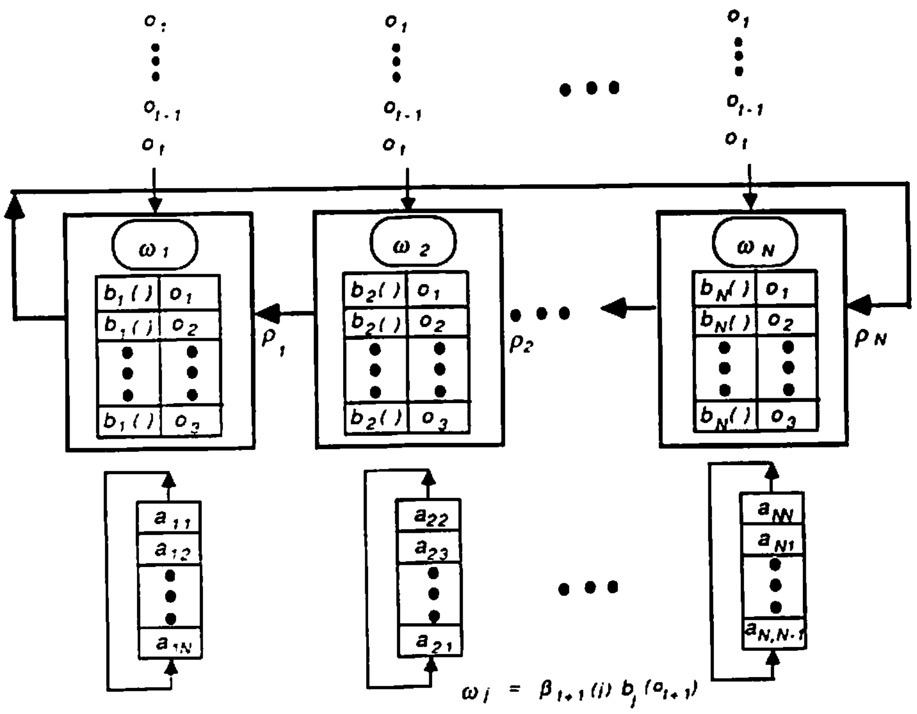


Figure 6.5: The execution of the backward evaluation procedure on the HMM systolic array.

$$Pr(O,\ I^*|\lambda) = \max_{\text{all } I} Pr(O,\ I|\lambda)$$

Because of the intimate similarity between the Viterbi algorithm (without backtracking steps) and the forward evaluation procedure, the HMM systolic array can again be easily adapted to the Viterbi algorithm.

1. **Initialization:** Each PE, say the i-th PE, receives π_i and o_0 from its corresponding input channel and computes its initial forward likelihood $\delta_0(i) = \pi_i\ b_i(o_0)$ in one clock unit. An initial state variable $\psi_0(i)$ is arbitrarily assigned zero value. The initial probability $\delta_0(i)$ is then multiplied by a_{ii} to form the product which is assigned to the comparator variable η_i. This initiates the iteration cycle at time $t = 1$ and $\delta_0(i)$ is then pipelined

leftward to be cycled through the HMM systolic array.

2. **Recursion:** Each PE, say the i-th PE, upon receiving $\delta_t(j)$ at each clock of the t-th cycle, will fetch the corresponding transition probability a_{ij} from the cyclic-shift memory and form the product $a_{ij}\delta_t(j)$. For each cycle, the two variables η_i and $\psi_{t+1}(i)$ are arbitrarily assigned a zero initial value. The product $a_{ij}\delta_t(j)$ is then compared with η_i (stored in the i-th PE) at each PE as follows

$$\text{if } \eta_i \geq a_{ij}\delta_t(j) \quad \text{then} \quad \begin{cases} \eta_i & \longleftarrow \eta_i \\ \psi_{t+1}(i) & \longleftarrow \psi_{t+1}(i) \end{cases}$$

$$\text{if } \eta_i < a_{ij}\delta_t(j) \quad \text{then} \quad \begin{cases} \eta_i & \longleftarrow a_{ij}\delta_t(j) \\ \psi_{t+1}(i) & \longleftarrow j \end{cases}$$

After N clock units, all the comparisons have been made, and the comparator variable η_i is multiplied by $b_i(o_{t+1})$ and the product is assigned to $\delta_{t+1}(i)$. Hence,

$$\begin{aligned} \delta_{t+1}(i) &= \eta_i b_i(o_{t+1}) \\ &= \max_{1 \leq j \leq N} [a_{ij}\delta_t(j)] b_i(o_{t+1}) \\ \psi_{t+1}(i) &= \arg\max_{1 \leq j \leq N} [a_{ij}\delta_t(j)] \end{aligned}$$

The newly generated $\delta_{t+1}(i)$ is again multiplied by a_{ii} to form the new η_i, and start a new recursion cycle. The variable η_i is then pipelined leftward through the HMM systolic array.

3. **Termination:** After T cycles have passed, each $\delta_T(i)$ is created at the i-th PE. One more cycle is required to find the maximum value of $\{\delta_T(i)\}$ among all the N PEs. This final cycle also starts the backtracking operations.

$$\begin{aligned} p^* &= \max_{1\le i\le N} [\delta_T(i)] \\ i_T^* &= \arg\max_{1\le i\le N} [\delta_T(i)] \end{aligned}$$

Because of the sequential nature of the backtracking processes, we would like to perform the backtracking operations by using an appended PE connected to the leftmost PE of the HMM systolic array as shown in Figure 6.6, so that the pipelinability of the whole system won't be influenced. By using this appended PE, all the original PEs can still be used to perform the Viterbi algorithm on the next observation sequence at the same time that all the $\{v_T(i)\}$ being compared at the appended PE. Because of the existence of the appended PE, we need to send all the $\{\psi_t(i)\}$ values created in the i-th PE at the t-th cycle to the appended PE via pipelining at $(t+1)$-th cycle.

4. **Backtracking:** Once the i_T^* is produced at the appended PE during the extra time cycle, and all the $\{\psi_t(i)\}$ have been sent to and stored in the appended PE, the backtracking for the optimal state sequence can be started in a straightforward sequential way.

$$i_t^* = \psi_{t+1}(i_{t+1}^*)$$

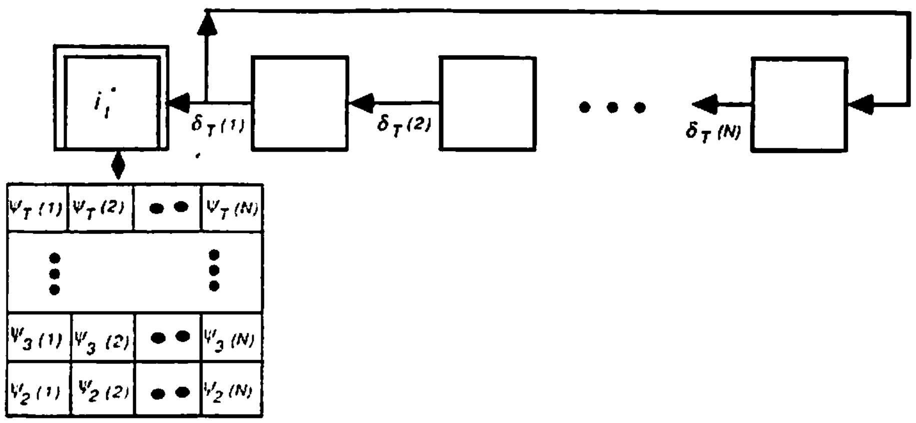


Figure 6.6: The execution of the Viterbi algorithm on the HMM systolic array to find the most probable state sequence.

6.2.2 Array Design for the Learning Phase of an HMM

The HMM systolic array can also be used to implement the Baum-Welch reestimation algorithm in a fully pipelined fashion. Similar to the systolic implementation of the BP learning algorithm, we can schedule the overall operations in the learning phase as follows.

1. Given the training sequence O, perform the forward evaluation procedure (like the forward step in BP learning) on the HMM systolic array as discussed in Section 6.2.1 (see Eqs. 6.6, 6.7, 6.8).

2. After the scoring probability $Pr(O|\lambda)$ is calculated, the backward evaluation procedure is started as discussed in Section 6.2.1 (see Eqs. 6.9, 6.10).

3. Along with the recursion of the backward evaluation procedure, the working variables $\{\gamma_t(i)\}$, $\{\xi_t(i,j)\}$ are computed cycle-by-cycle (see Eqs. 6.15, 6.16).

4. At the end of the backward evaluation procedure, the model parameters (A, B, π) are updated (see Eqs. 6.17, 6.18, 6.19).

Repeat 1 to 4 until convergence.

Now let us assume the forward evaluation procedure is completed and all the forward likelihoods $\{\alpha_t(i),\ t = 0,\ 1,\ \ldots,\ T\}$ are all computed and stored in sequence at the i-th PE. Three extra working variables at i-th PE need to be defined, $\{bb_i(k),\ k = 1,\ 2, \ldots, M\}$, $\{aa_{ij},\ i = 1,\ 2, \ldots, N\}$, and bt_i. These three extra working variables and the two previously defined working variables $(\{\gamma_t(i)\}, \{\xi_t(i,j)\})$ will perform the following operations at the same time that the backward evaluation procedure is performing its t-th cycle operations (see Figure 6.7).

1. Whenever a new $\beta_t(i)$ is created at the i-th PE (note that $\alpha_t(i)$ is assumed to be stored in the i-th PE).

$$\gamma_t(i) = \alpha_t(i)\ \beta_t(i)$$

2. Again the fast associative retrieval mechanism is used to increment one of the working variables $\{bb_i(k)\}$,

$$bb_i(k) \longleftarrow \begin{cases} bb_i(k) + \gamma_t(i) & \text{if } o_t = k \\ bb_i(k) & \text{if } o_t \neq k \end{cases}$$

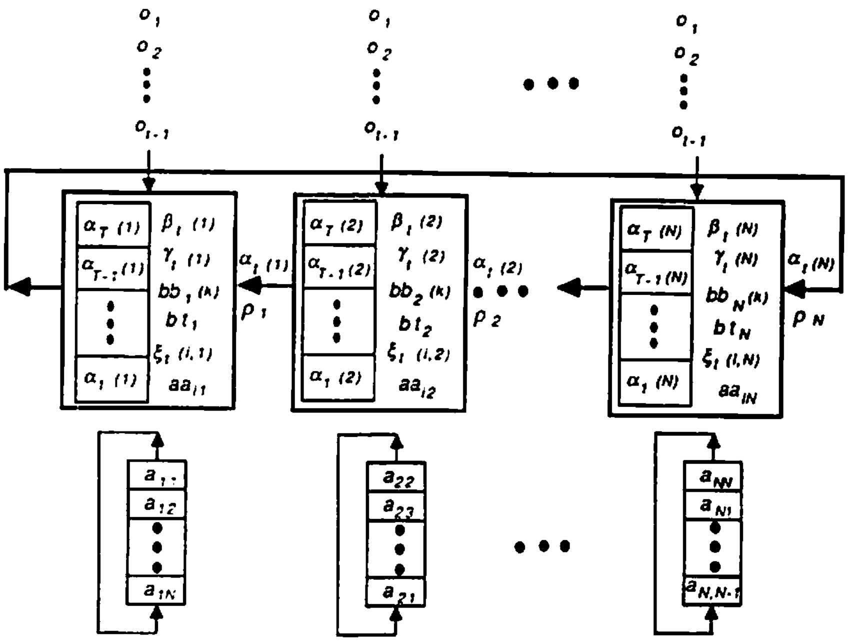


Figure 6.7: The execution of the learning phase on the HMM systolic array, which combines both the operations of the backward evaluation procedure and reestimation algorithm at the same time.

3. The normalization variable bt_i needs to be incremented also,

$$bt_i = bt_i + \gamma_t(i)$$

4. While calculating $\beta_t(j)$, we pre-compute the product $\beta_{t+1}(i)b_i(o_{t+1})a_{ij}$ at the i-th PE (which is to be added to the cycling accumulator ρ_j). This product is now multiplied by $\alpha_t(j)$ and the product assigned to the modified working variable $\xi_t(i,j)$.

$$\xi_t(i,j) = \alpha_{t-1}(j)\ b_i(o_t)\ \beta_t(i)\ a_{ij}$$

Along with the cycling of ρ_j, the forward likelihood $\alpha_t(j)$ needs to be cycled in the HMM systolic array to ensure the multiplication can be carried out in the i-th PE at the right time.

5. After $\xi_t(i,j)$ is computed, the working variable aa_{ij} in the i-th PE can thus be incremented.

$$aa_{ij} \longleftarrow aa_{ij} + \xi_t(i,j).$$

After the recursions of the backward evaluation procedure are completed, the three extra working variables are all assigned their proper values.

$$\begin{aligned} bt_j &= \sum_{t=0}^{T} \gamma_t(j) \\ bb_j(k) &= \sum_{t=0,\ o_t=k}^{T} \gamma_t(j) \\ aa_{ij} &= \sum_{t=1}^{T} \xi_t(i,j) \end{aligned}$$

At the end, an additional cycle is required to perform the parameter updatings based on Eqs. 6.17, 6.18, 6.19,

$$\bar{a}_{ij} = aa_{ij}/(bt_j - \gamma_T(j)) \tag{6.31}$$

$$\bar{b}_j(k) = bb_j(k)/bt_j \tag{6.32}$$

$$\bar{\pi}_i = \gamma_1(i)/Pr(O|\lambda). \tag{6.33}$$

6.2.3 Systolic Array for Left-to-Right HMMs

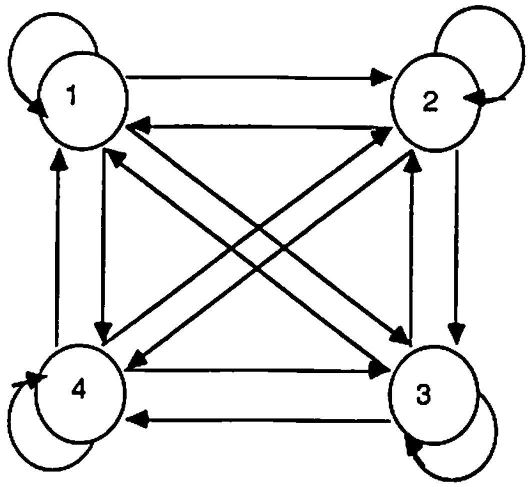
In the above discussions, we are only concerned about the general HMM, which is assumed to have a full state transition matrix, i.e., all the $\{a_{ij}\}$ in the state transition matrix **A** are nonzero entries (see Figure 6.8(a)). For some applications, we are interested in models where constraints are imposed on the HMM. One popular model is the *left-to-right* HMM as shown in Figure 6.8(b). A left-to-right HMM has a upper triangular state transition matrix. For a six state ($N = 6$) and bandwidth ($B = 2$) HMM, **A** can be expressed as:

$$\mathbf{A} = \begin{bmatrix} a_{11} & 0 & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & 0 & 0 & 0 \\ 0 & 0 & a_{43} & a_{44} & 0 & 0 \\ 0 & 0 & 0 & a_{54} & a_{55} & 0 \\ 0 & 0 & 0 & 0 & a_{65} & a_{66} \end{bmatrix} \tag{6.34}$$

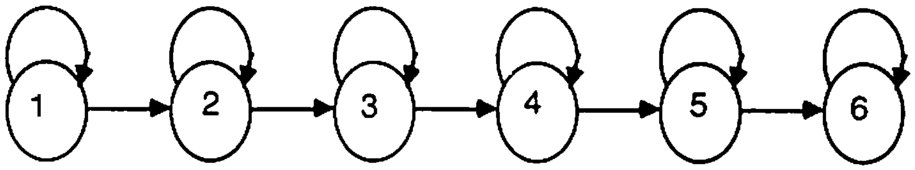
The left-to-right HMM, which is widely used in isolated word recognition tasks, inherently imposes a temporal ordering since the lower numbered states account for observations occurring prior to those of higher numbered states [134]. In isolated word recognition, a word utterance has an unambiguous progression through the state sequence, and the number of states needed for each word model is usually manageable.

Mapping Left-to-Right HMMs to a Bidirectional Systolic Array

The previously proposed HMM systolic array can be effectively adapted (by adding a few semi-global links) to exploit the banded (or circularly banded) structure of the state transition matrix [99]. The modified cascaded DG for the ergodic HMM ($B = 2$) defined in Eq. 6.34 is shown in Figure 6.9(a). The same projection and scheduling vectors will lead us to the bidirectional HMM systolic



(a)



(b)

Figure 6.8: (a) A general (ergodic) HMM, in which each state in this model is possible to reach any states (including itself). (b) A left-to-right HMM, in which the lower numbered state always precedes a higher numbered state.

array with global interconnection links (of length $B-1$) as shown in Figure 6.9(b). Note that the newly generated forward likelihoods $\{\alpha_t(i)\}$ are circularly shifted rightward by $(B-1)$ positions via semi-global wires. The bidirectional HMM systolic array can be used to process the left-to-right HMM, with a factor of N/B savings in processing time. This savings is very significant for most applications. Basically, the semi-global interconnection links are only used to reposition the propagating data. The implementation of the backward evaluation procedure and the reestimation algorithm of the learning phase of the left-to-right HMM on the bidirectional HMM systolic array is straightforward. Basically, an analysis similar to that of the general HMM can be carried through for the left-to-right HMM. Only small modifications to the semi-global links for the repositioning of the propagating values are required.

6.2.4 Incorporating the Scaling Scheme

It is observed that both $\alpha_t(i)$ and $\beta_t(i)$ tend to zero with geometric speed due to the consecutive multiplications of probability values which are less than 1.0. In order to avoid the *mathematical underflow* problem frequently encountered in the learning phase, a scaling scheme using a normalization technique has been proposed [104,67]. Although this scheme can successfully overcome the mathematical underflow problems, the parallel implementation of this scheme is hindered by its sequential nature.

A slightly modified version of the original scaling scheme (see [104,67]) is proposed, which can successfully overcome the problems of mathematical underflow,

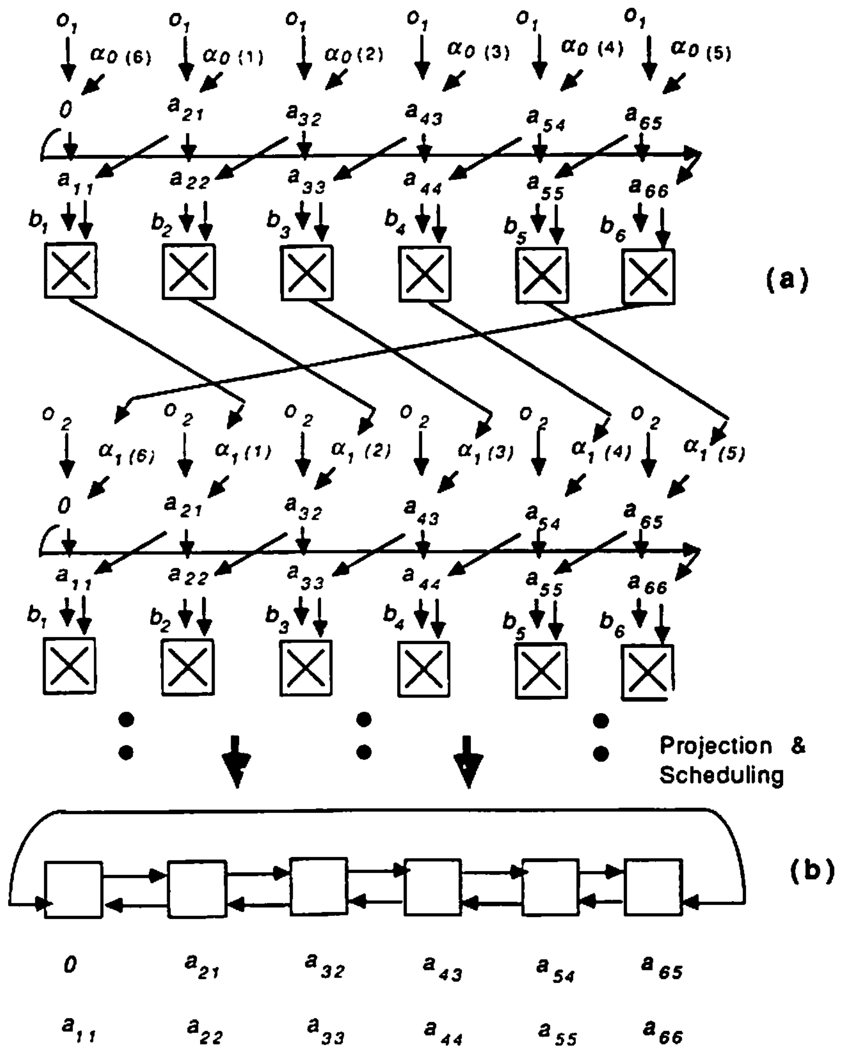


Figure 6.9: (a) The DG for the left-to-right HMM with $N = 6$ and $B = 2$. (b) The bidirectional HMM systolic array with global interconnection links (of length $B - 1$).

yet be implemented on the HMM systolic array without degrading the pipelining rate.

Scaling in Forward Evaluation The modified forward likelihood $\{\alpha_t'(i)\}$ (see Eq. 6.7), which can prevent underflow, are defined as [67]

$$\alpha_{t+1}''(i) = [\sum_{j=1}^{N} a_{ij}\ \alpha_t'(j)]\ b_i(o_{t+1}) \tag{6.35}$$

$$\begin{aligned} \alpha_{t+1}'(i) &= \frac{\alpha_{t+1}''(i)}{\phi_t} \\ &= \frac{\alpha_{t+1}''(i)}{\sum_{i=1}^{N} \alpha_t'(i)} \\ &= \frac{\alpha_{t+1}(i)}{\sum_{i=1}^{N} \alpha_t(i)} \\ &= \frac{\alpha_{t+1}(i)}{\prod_{\tau=1}^{t} \varphi_\tau} \end{aligned} \tag{6.36}$$

where

$$\begin{aligned} \phi_t &= \sum_{i=1}^{N} \alpha_t'(i) \\ &= \frac{\sum_{i=1}^{N} \alpha_t(i)}{\sum_{i=1}^{N} \alpha_{t-1}(i)} \end{aligned} \tag{6.37}$$

Note that $\phi_0 = 1$, and

$$\prod_{\tau=1}^{T} \varphi_\tau = \sum_{i=1}^{N} \alpha_T(i) = Pr(O|\lambda) \tag{6.38}$$

The HMM systolic array can be easily adapted to recursively calculate the modified forward likelihood $\{\alpha_{t+1}'(i)\}$ in a fully pipelined fashion. As discussed in

Section 6.2.1, at the t-th cycle, the HMM systolic array requires N clock units to calculate $\alpha''_{t+1}(i)$ at each PE, i.e., the i-th PE, (see Eq. 6.35). At the same time, when $\alpha'_t(i)$ is cycling in the ring array to accumulate the required components to form $\alpha''_{t+1}(i)$ at the i-th PE, the computations of ϕ_t shown in Eq. 6.36 take place. When $\alpha'_t(i)$ comes back to the i-th PE after N clock units, the accumulations of ϕ_t are completed, and we are ready to compute $\alpha'_{t+1}(j)$ (see Eq. 6.36). Note that $\{\phi_t\}$ will all be stored at each PE (instead of storing $\{\alpha'_{t+1}(i)\}$ only) for use in the learning phase.

Scaling in the Backward Evaluation and Reestimation In a similar manner, the modified backward likelihood $\{\beta'_t(i)\}$ (see Eq. 6.10) can also be defined as [67]:

$$\beta''_t(i) = \sum_{j=1}^{N} \beta'_{t+1}(j)\, b_j(o_{t+1})\, a_{ji} \tag{6.39}$$

$$\begin{aligned} \beta'_t(i) &= \frac{\beta''_t(i)}{\phi_t} \\ &= \frac{\beta_t(i)}{\prod_{\tau=t}^{T} \phi_\tau} \end{aligned} \tag{6.40}$$

Since all the $\{\phi_t\}$ are stored at each PE, the calculations of $\{\beta'_t(i)\}$ nearly follow the presentation in Section 6.2.1.The only additional computation is the division operation of ϕ_t.

It can be easily shown that two modified working variables, $\gamma'_t(i)$ and $\xi'_t(i,j)$ can be derived from $\{\alpha'_t(j)\}$ and $\{\beta'_t(j)\}$ as follows.

$$
\begin{aligned}
\gamma_t'(i) &= Pr(i_t = q_i | O, \lambda) \\
&= \frac{Pr(i_t = q_i, O|\lambda)}{Pr(O|\lambda)} \\
&= \frac{\alpha_t(i)\beta_t(i)}{Pr(O|\lambda)} && (6.41) \\
&= \alpha_t'(i)\beta_t'(i) && (6.42)
\end{aligned}
$$

$$
\begin{aligned}
\xi_t'(i,j) &= Pr(i_t = q_i, i_{t-1} = q_j | O, \lambda) \\
&= \frac{Pr(i_t = q_i, i_{t-1} = q_j, O|\lambda)}{Pr(O|\lambda)} \\
&= \frac{a_{ij}\ \alpha_{t-1}(j)\ b_i(o_t)\ \beta_t(i)}{Pr(O|\lambda)} \\
&= \frac{a_{ij}\ \alpha_{t-1}'(j)\ b_i(o_t)\ \beta_t'(i)}{\phi_{t-1}} && (6.43)
\end{aligned}
$$

These variables can be computed in the same manner as $\gamma_t(i)$ and $\xi_t(i,j)$ discussed in Section 6.2.2. (Note that the scaling constant ϕ_{t-1} is available at all the PEs at the moment of computing $\xi_t'(i,j)$).

By using the modified forward/backward variables in the computations of the learning phase, all the procedures discussed in Section 6.2.2 are still valid, except that the normalization operations in Eq. 6.33 can be avoided.

Chapter 7

Future Research

The research of ANNs involves a very broad spectrum of disciplines, including algorithm analyses, application understanding, parallelism extractions, array architectures, programming techniques, functional primitives, structural primitives, and the numerical performance of algorithms. It is also important to compare the neural nets approach with the other conventional methods [109], e.g. simulated annealing [72], hidden Markov model [134,165], pattern recognition [123,128], and nonlinear programming [28].

Although the applicational and architectural studies will play a very important role in ANN research, from our point of view, algorithmic studies are the most important phase in an integrated ANN research effort. At this stage of neural nets research, a lot of results are simply verified by simulations without theoretical or mathematical basis. There are various mathematical aspects in the algorithmic studies of ANNs worthy of our immediate pursuits, e.g., expressibility and discrimination capabilities, generalization capabilities, convergence in the retrieving and learning phases. and the merging of HMMs and ANNs.

7.1 Expressibility and Discrimination of ANNs

It is suggested in [103] that the ability of Perceptron-like multilayer nets to infer the *inherent rule* is a kind of real-valued function interpolation. The neural net approximates an arbitrary function by using weighted sum of various sigmoid functions, i.e.. weighted sum of some basis functions "bumps," which is analogous to the method of splines for approximating arbitrary functions [15]. Just like splines, the *tanh* functions are constructed in such a way to maximize smoothness. Approximating a function with sums of sigmoids is also similar to a mode decomposition of the function in terms of a certain orthogonal function basis set, e.g., Fourier functional, principal component, or eigenvalue-eigenvector analysis. Let us consider a two layer net with N_0 input units and N_1 hidden units. The activation value $a_i^{(2)}$ of the i-th neuron at the output layer can be expressed as

$$a_i(2) = f\{\sum_{k=1}^{N_1} w_{ik}(2) f[\sum_{j=1}^{N_0} w_{kj}(1) a_j(0) + \theta_k(1)] + \theta_i(2)\}$$

where $f[\sum_{j=1}^{N_0} w_{kj}(1) a_j(0) + \theta_k(1)]$ can be regarded as one type of basis function and $w_{ik}(2)$ determines the weights. Using the terminology of Fourier analysis in electromagnetic theory, the neural network can adjust the proper wavenumbers of the basis functions by changing $\{w\}$. The basis functions and weights exhibit a similar relationship corresponding to that of eigenvectors and eigenvalues in a principal component analysis.

Given a fixed activation function (e.g., sigmoid), a multilayer Perceptron has an inherent limitation on how accurately it may express the desired function. In other words, there exists an inevitable gap between the target model and the closest model representable by the ANNs.

7.2 Generalization Capability of ANNs

The most prominent weakness of neural architecture is its poor generalization [34]. The problem of generalizing the classification on a set of training patterns to the classification of a new test pattern unobserved before is the most important goal of today's ANN research. In fact, given a finite sets of training I/O pairs, it would be impossible for any learning algorithm to yield the exact generalization. Obviously, the sample set of learning situations will affect the resulting characteristics of the trained neural system. However, a good trainer should be able to guarantee that the generalization will asymptotically converge to give the most representable output (of this learning algorithm) given long enough training data.

The generalization capability of layered linear threshold network has been studied based on the *dichotomy* analysis [30,123,17]. In these methods, the generalization capability is analyzed based on the probability that a new test pattern can be categorized into the originally separated regions without introducing ambiguity. There is still a big gap between these learning-rule independent theoretic results and the specific neural models.

Also studied in [148] is the generalization capability of multilayer Perceptron. A special example of on the so-called 3-in-8 problem is used, in which the net responds a "1" if there are at least 3 adjacent active bits in an 8 bits input sequence are detected; otherwise the net responds a "0." It is reported that the back-propagation indeed realizes an order 3 AND-OR predicate of the 3-in-8 problem. For small size of training patterns, the network learns the discrimination by rote and takes no advantage of the regularity of the task. When the size of the training patterns reach some ratio of that of the overall allowed training patterns, the net seems to switch from rote learning to more "intelligent" behavior. At this time, some regularity is established in the internal representation of the hidden units and there is no use to increase the number of hidden units.

7.3 Convergence Issues in the Retrieving and Learning Phases

Two main difficulties exist in using current ANNs for searching for a *valid* solution in an optimization problem: one is the potential trap at local minimum; the other is that a direct minimizing solution may not satisfy the given constraints. This leads us to the research in the convergence of the retrieving phase. Possible techniques have been proposed to incorporate into the mechanisms of ANNs, e.g., simulated annealing approaches [153], or Lagrange multiplier techniques [129]. Other issues are the convergence in the presence of the discretization and the parallel updating of the system dynamics in the single layer feedback system. These issues are of importance in the digital parallel implementation. Another important concern is the convergence properties due to the initial state of a neural net.

The convergence issues in the learning phase play an important role in the performance of the specific learning algorithm. For example, as we discussed in Chapter 3, by optimal selection of hidden unit size and learning rate, the convergence can be speedup significantly. There are some other methods that might be useful. For example, the variable step size algorithm [46], or the block nonlinear projection methods [136,137].

7.4 The Merging of Multilayer Perceptrons and HMMs

In Chapter 6, we have presented a unifying viewpoint between a multilayer Perceptron and a hidden Markov model. It is observed that the hidden Markov models can be regarded as a special configuration of the multilayer Perceptron with a squashing type of nonlinear activation function. It is also shown that, through proper redefinition, the iterative gradient descent approach, which was used to derive the back-propagation learning in the multilayer Perceptron, can be successfully used to derive the Baum-Welch reestimation formulation in the hidden Markov models. Hopefully, through better understanding of the close interaction between these two models, deep insights toward the better usage and modification of these two algorithms can be discovered.

Bibliography

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

[2] H.M. Ahmed. Alternative arithmetic unit architectures for VLSI digital signal processors. In *VLSI and Modern Signal Processing*, Editors, S.Y. Kung et al, Chapter 16, pages 277–306, Prentice Hall, Inc., Englewood Cliffs, NJ, 1985.

[3] L.A. Akers, M.R. Walker, D. K. Ferry, and R. O. Grondin. Limited interconnectivity in synthetic systems. In *Neural Computers, Computer and Systems Science Series*, Editors, R. Eckmiller and C. V. D. Malsburg, pages 407–416, Springer-Verlag Inc., 1988.

[4] J. Alspector and R. B. Allen. A neuromorphic VLSI learning systems. In *Advanced Research on VLSI*, Editor, P. Losleben, pages 313–349, MIT Press, 1987.

[5] J. Alspector, R. B. Allen, V. Hu, and S. Satyanarayana. Stochastic learning networks and their electronic implementation. In *Proc. IEEE Conf. on Neural Information Processing Systems – Natural and Synthetic, Denver*, November 1987.

[6] S. I. Amari. Characteristics of randomly connected threshold-element network systems. *Proceedings of the IEEE*, 59:35–47, January 1971.

[7] S. I. Amari. A method of statistical neurodynamics. *Kybernetik (Biological Cybernetics)*, 14:201–225, 1974.

[8] J. A. Anderson. *Neurocomputing – Paper Collections.* MIT Press, 1988.

[9] J. A. Anderson. A simple neural network generating an interactive memory. *Mathematical Biosciences*, 14:197–220, 1972.

[10] J. A. Anderson, J. W. Silverstein, S. A. Ritz, and R. S. Jones. Distinctive features, categorical perception, and probability learning: some application of a neural model. *Psych. Rev.*, 84:413–451, 1977.

[11] J. A. Anderson, J. W. Silverstein, S. A. Ritz, and R. S. Jones. A memory storage model utilizing spatial correlation functions. *Kybenetik (Biological Cybernetics)*, 5, 1968.

[12] L. E. Atlas, T. Homma, and R. J. Marks II. A neural network model for vowel classification. In *Proc. IEEE ICASSP'87, Dallas*, 1987.

[13] L. R. Bahl, F. Jelinek, and R. L. Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 5(2):179–190, March 1983.

[14] D. H. Ballard, G.E. Hinton, and T. J. Sejnowski. Parallel visual computation. *Nature*, 306, 1983.

[15] D.H. Ballard and C.M. Brown. *Computer Vision*. Prentice Hall, Inc., N.J., 1982.

[16] C. R. Baugh and B. A. Wooley. A two's complement parallel array multiplication algorithm. *IEEE Trans Computers*, C-22:1045–1047, December 1973.

[17] E. B. Baum. On the capability of multilayer Perceptron. Technical Report, Jet Prop. Lab., Calif. Insti. of Technology, CA, 1987.

[18] E. B. Baum, J. Moody, and F. "'lczek. Internal representations for associative memory. In *Proc. IEEE Conf. on Neural Information Processing Systems – Natural and Synthetic, Denver*, November 1987.

[19] L.E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic function of a Markov process. *Inequalities*, 3:1–8, 1972.

[20] L.E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Ann. Math. Statistic.*, 41:164–171, 1970.

[21] R. R. Bitmead and B. D. O. Anderson. Exponentially convergent behavior of simple stochastic adaptive estimation algorithms. In *Proc. 17-th IEEE Conf. Decision and Control*, pages 580–585, 1979.

[22] H. Bourlard. A link between Markov models and multilayer Perceptrons? Presented in Int'l Neural Network Society (INNS), First Annual Meeting, Boston, September 1988.

[23] H. Bourlard and C. J. Wellekens. Multilayer Perceptrons and automatic speech recognition. In *Proc. IEEE 1st Intl' Conf. on Neural Networks, San Diego*, pages IV 407- IV 416, June 1987.

[24] D. J. Burr. Experiments with a connectionist text reader. In *Proc. IEEE 1st Intl' Conf. on Neural Networks, San Diego*, pages IV 717- IV 724, 1987.

[25] G. A. Carpenter and S. Grossberg. The art of adaptive pattern recognition by a self-organizing neural network. *IEEE Computer Magazine*, 21:77–88, March 1988.

[26] G. A. Carpenter and S. Grossberg. Art2: self-organization of stable category recognition codes for analog input patterns. In *Proc. IEEE 1st Intl' Conf. on Neural Networks, San Diego*, pages II 727- II 736, 1987.

[27] Y. Censor. Row-action methods for huge and sparse systems and their applications. *SIAM Review*, 23:444–466, October 1981.

[28] L. O. Chua and G. N. Lin. Nonlinear programming without computation. *IEEE Trans. on Circuits and Systems*, 31:182–188, Feburary 1984.

[29] M. A. Cohen and S. Grossberg. Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Trans. on Systems, man and Cybernectics*, 13(5):815–825, September 1983.

[30] T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Trans. on Electronic Computers*, 14:326–334, 1965.

[31] J. D. Cowan and D. H. Sharp. Neural nets. Technical Report, Mathematical Dept., University of Chicago, 1987.

[32] J-M Delosme and I. C. F. Ispen. Efficient systolic arrays for the solution of Toeplitz systems: an illustration of a methodology for the construction of systolic architectures in VLSI. *International Workshop on Systolic Arrays, University of Oxford*, F2, July 1986.

[33] E. Deprettere, P. Dewilde, and P. Udo. Pipelined CORDIC architecture for fast VLSI filtering and array processing. In *Proc. ICASSP'84*, pages 41.A.6.1 - 41.A.6.4, 1984.

[34] R. Eckmiller and C. V. D. Malsburg. *Neural Computers*. NATO ASI Series F, Computer and System Science, Springer-Verlag, 1987.

[35] G. Eichmann and H. J. Gaulfield. Optical learning (inference) machines, vol. 24. *Applied Optics*, 1985.

[36] N. H. Farhat, D. Psaltis, A. Prata, and E. Paek. Optical implementation of the Hopfield model. *Applied Optics*, 24:1469–1475, May 1985.

[37] A. D. Fisher and J. N. Lee. Optical associative processing elements with versatile adaptive learning capability. In *Proc. IEEE COMPCOM Meeting*, 1985.

[38] K. Fukushima. Cognitron: a self-organizing multilayered neural network. *Biological Cybernetics*, 20:121–136, 1975.

[39] K. Fukushima. Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193-202, April 1980.

[40] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 6:721–741, November 1984.

[41] R. P. Gorman and T. J. Sejnowski. Learned classification of sonar targets using a massively parallel network. *IEEE Trans. on ASSP*, 36:1135–1140, July 1988.

[42] H. P. Graf and P deVegvar. A CMOS implementation of a neural network model. In *Advanced Research on VLSI*, Editor, P. Losleben, pages 351–367, MIT Press, 1987.

[43] H. P. Graf, W. Hubbard, L. D. Jackel, and P. G. N. DeVegvar. A CMOS associative memory chip. In *Proc. IEEE 1st Intl' Conf. on Neural Networks, San Diego*, pages III 461- III 468, 1987.

[44] S. Grossberg. Adaptive pattern classification and universal recoding: Part I, parallel development and coding of neural feature detectors. *Biological Cybernetics*, 23:121–134, 1976.

[45] F. Guenther and C. Briggs. Bit-level pipelined VLSI implementation of the CORDIC algorithm. Class Report, Department of Electrical Engineering, EE 539, Princeton University, May 1987.

[46] R. W. Harris, D. M. Chabries, and F. A. Bishop. A variable step (vs) adaptive filter algorithm. *IEEE Trans. on ASSP*, 34:309–316, April 1986.

[47] D. O. Hebb. *The Organization of Behavior.* Wiley Inc., New York, 1949.

[48] R. Hecht-Nielsen. Performance limits of optics, electro-optics, and electronic neurocomputers. In *Proc. SPIE, Optical and Hybrid Computing*, 1987.

[49] W. D. Hillis. *The Connection Machine.* Massachusetts Institute Technology Press, 1985.

[50] G. E. Hinton. Connectionist learning procedure. Technical Report CMU-CS-87-115, Carnegie Mellon University, September 1987.

[51] 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)*, Chapter 7, Editors, D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, pages 282–317, MIT Press, Cambridge, Massachusetts, 1986.

[52] J. J. Hopfield. Neural network and physical systems with emergent collective computational abilities. In *Proc. Natl'. Acad. Sci. USA*, pages 2554–2558, 1982.

[53] J. J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proc. Natl.. Acad. SCi. USA*, 81:3088–3092, 1984.

[54] J. J. Hopfield and D. W. Tank. Neural computation of decision in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.

[55] W. Y. Huang and R. P. Lippmann. Neural net and traditional classifiers. In *Proc. IEEE Conf. on Neural Information Processing Systems - Natural and Synthetic, Denver*, November 1987.

[56] J. Hutchinson, C. Koch, J. Luo, and C. Mead. Computing motion using analog and binary resistive networks. *IEEE Computer Magazine*, 21:52–63, March 1988.

[57] J. N. Hwang and S. Y. Kung. A unifying viewpoint between multilayer Perceptrons and hidden Markov models. Submitted to *IEEE Int'l Symposium on Circuits and Systems, ISCAS'89, Portland*, May 1989.

[58] J. N. Hwang, J. A. Vlontzos, and S.Y. Kung. Parallel architectures and implementation considerations for hidden Markov model. In *Proc., SPIE, Visual Comm. and Image Processing III, Cambridge, Massachusetts*, November 1988.

[59] J. N. Hwang, J. A. Vlontzos, and S.Y. Kung. Parallel architectures and implementation considerations for hidden Markov model. Accepted by *IEEE Trans. on Acoustics, Speech, and Signal Processing*, March 1988.

[60] K. Hwang and Joydeep Ghosh. Hypernets for parallel processing with connectionist architectures. Technical Report CRI-87-03, University of Southern California, January 1987.

[61] W.C. Ridgway III. An adaptive logic system with generalizing properties. Ph.D. Thesis, Stanford Electronics Labs., Rep. 1556-1, Stanford University, April 1962.

[62] F. Jelinek. The development of an experimental discrete dictation recognizer. *Proceedings of the IEEE*, 73(11):1616–1624, November 1985.

[63] C. Jorgensen, W. Hamel, and C. Weisbin. Autonomous robot navigation. *Byte*, 223–235, January 1986.

[64] C. C. Jorgensen. Neural network representation of sensor graphs for autonomous robot navigation. In *Proc. IEEE 1st Intl' Conf. on Neural Networks, San Diego*, pages IV 507- IV 515, June 1987.

[65] N. H. Brown Jr. Neural network implementation approaches for the connection machine. In *Proc. IEEE Conf. on Neural Information Processing Systems - Natural and Synthetic, Denver*, November 1987.

[66] B. H. Juang. On the hidden Markov model and dynamic time warping for speech recogintion - a unified view. *AT&T Bell Laboratories Technical Journal*, 63(7):1213–1243, September 1984.

[67] B. H. Juang and L. R. Rabiner. Mixture autoregressive hidden Markov models for speech signals. *IEEE Trans. on ASSP*, 33(6):1404–1413, December 1985.

[68] S. Kaczmarz. Angenäherte auflösung von systemen linearer gleichungen. *Bull. Int. Acad. Polon. Sci. (Series A)*, 355–357, 1937.

[69] R. M. Karp, R. E. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. *Journal of ACM*, 14(3):563–590, July 1967.

[70] M. Kawato, Y. Uno, and M. Isobe. A hierarchical model for voluntary movement and its application to robotics. In *Proc. IEEE 1st Intl' Conf. on Neural Networks, San Diego*, pages IV573- IV582, June 1987.

[71] S. Kayalar and H. L. Weinert. Error bounds for the method of alternating projections. *Mathematics of Control, Signals, and Systems*, 1:43–59, 1988.

[72] S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecci. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[73] C. Koch, J. Marroquin, and A. Yuille. Analog neuronal networks in early vision. *Proc. of National Academy Science*, 83:4263–4267, 1986.

[74] T. Kohonen. *Associative Memory - A System-Theoretic Approach.* Springer-Verlag, New York, 1977.

[75] T. Kohonen. Correlation matrix memories. *IEEE Transactions on Computer*, Vol. C-21, 1972.

[76] T. Kohonen. *Self-Organization and Associative Memory, Series in Information Science, Vol. 8.* Springer-Verlag, New York, 1984.

[77] T. Kohonen. Self-organized formation of topologically correct feature map. *Biological Cybernetics*, 43:59–69, 1982.

[78] B. Kosko. Adaptive bidirectional associative memory. *Appl. Opt.*, 1987.

[79] H.T. Kung. Why systolic architectures? *IEEE Computer*, 15(1), January 1982.

[80] H.T. Kung and C.E. Leiserson. Systolic arrays (for VLSI). In *Sparse Matrix Symposium*, pages 256–282, SIAM, 1978.

[81] S. Y. Kung. *VLSI Array Processors.* Prentice Hall Inc., N.J., 1988.

[82] S. Y. Kung, J. N. Hwang, and S. C. Lo. Mapping digital signal processing algorithms onto VLSI systolic/wavefront arrays. In *Proc. 20th Annual Asilomar Conf. on Signals, Systems and Computers*, pages 6–12, November 1986.

[83] S. Y. Kung, S. N. Jean, S.C. Lo, and P. S. Lewis. Design methodologies for systolic arrays: mapping algorithms to architectures. In *Chapter 6, Signal Processing Handbook*, pages 145–191, Marcel Dekker Inc., 1988.

[84] S. Y. Kung, P. S. Lewis, and S. N. Jean. Canonic and generalized mapping from algorithms to arrays - a graph based methodology. In *Proc. of the Hawaii Inter. Conf. on System Sciences*, pages 124–133, January 1987.

[85] S. Y. Kung, P. S. Lewis, and S. C. Lo. On optimally mapping algorithms to systolic arrays with application to the transitive closure problem. In *Proc. IEEE Int. Sym. on Circuits and Systems*, pages 1316–1322, 1986.

[86] S. Y. Kung, S. C. Lo, S. N. Jean, and J. N. Hwang. Wavefront array processors: from concept to implementation. *IEEE Computer Magazine*, 18–33, July 1987.

[87] S. Y. Kung, S. C. Lo, and P. S. Lewis. Optimal systolic design for the transitive closure and the shortest path problems. *IEEE Trans. on Computer*, C-36:603–614, May 1987.

[88] S. Y. Kung, R. W. Stewart, and S. C. Lo. Systolic applications for digital image/video processing applications. In *Proc. SPIE'87, Real Time Image Processing*, November 1987.

[89] S.Y. Kung. On supercomputing with systolic/wavefront array processors. *Proceedings of the IEEE*, 72(7), July 1984.

[90] S.Y. Kung. Parallel architectures for artificial neural nets. In *Proc. IEEE Int'l Conf. on Systolic Array, San Diego*, pages 163– 174, May 1988.

[91] S.Y. Kung, K.S. Arun, R.J. Gal-Ezer, and D.V. Bhaskar Rao. Wavefront array processor: language, architecture, and applications. *IEEE Transactions on Computers, Special Issue on Parallel and Distributed Computers*, C-31(11):1054–1066, Nov 1982.

[92] S. Y. Kung and J. N. Hwang. An algebraic projection analysis for optimal hidden units size and learning rate in back-propagation learning. In *Proc. IEEE Intl' Conf. on Neural Networks, San Diego*, pages I 363– I 370, July 1988.

[93] S.Y. Kung and J.N. Hwang. Digital VLSI architectures for artificial neural nets. In *Neural Networks for Computing, Snowbird, Utah*, April 1988.

[94] S.Y. Kung and J.N. Hwang. An efficient triarray systolic design for real-time Kalman filtering. In *Proc. IEEE ICASSP'88, New York*, pages 2045–2048, April 1988.

[95] S.Y. Kung and J.N. Hwang. Neural network architectures for robotic processing. To appear in IEEE Journal of Robotics and Automation, special issue on Computational Algorithms and Architectures in Robotics and Automation, 1988.

[96] S. Y. Kung and J. N. Hwang. Parallel architectures for artificial neural san diegnets. In *IEEE Int'l Conf. on Neural Networks, San Diego*, pages II 165– II 172, July 1988.

[97] S.Y. Kung and J.N. Hwang. Systolic architectures for Kalman filtering. In *Proc. 21st Annual Asilomar Conf. on Signals, Systems and Computers*, November 1987.

[98] S.Y. Kung and J.N. Hwang. Systolic array designs for Kalman filtering. Accepted by IEEE Trans. on Acoustics, Speech, and Signal Processing, October 1988.

[99] S.Y. Kung and J.N. Hwang. Systolic design for state space models: Kalman filtering and neural networks. In *26th IEEE Conf. on Decision and Control, Los Angeles*, pages 1461–1467, December 1987.

[100] S.Y. Kung, J.N. Hwang, and S.C. Lo. Systolic/wavefront arrays for image processing algorithms. In *Proc. MARI'87, Intelligent Networks and Machines*, May 1987.

[101] S.Y. Kung, J.N. Hwang, and S.W. Sun. Efficient modeling of multilayer feed-forward neural nets. In *Proc. IEEE ICASSP'88, New York*, pages 2160–2163, April 1988.

[102] S.Y. Kung and H. K. Liu. An optical inner-product array processor for associative retrieval. In *SPIE'86, Los Angeles*, January 1986.

[103] A. Lapedes and R. Farber. Nonlinear signal processing using neural networks. In *Proc. IEEE Conf. on Neural Information Processing Systems – Natural and Synthetic, Denver*, November 1987.

[104] S. E. Levinson, L. R. Rabiner, and M. M. Sondhi. An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition. *The Bell System Technical Journal*, 62:1035–1074, April 1983.

[105] P. S. Lewis. *Algorithms and Architectures for Adaptive Least Squares Signal Processing, with Applications in Magnetoencephalograph.* Ph.D. thesis, Dept. of Electrical Engineering, University of Southern California, August 1988.

[106] P. S. Lewis. Qr algorithm and array architectures for a multichannel adaptive least squares lattice filter. In *Proc. IEEE ICASSP'88, New York*, pages 2041–2044, April 1988.

[107] R. Linsker. Self-organization in a perceptual network. *IEEE Computer Magazine*, 21:105–117, March 1988.

[108] L. A. Liporace. Maximum likelihood estimation for multivariate observations of Markov sources. *IEEE Trans. on Information Theory*, 28:729–734, September 1982.

[109] R. P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP magazine*, 4:4–22, 1987.

[110] S. C. Lo. *Mapping Algorithms to VLSI Array Processors.* Ph.D. thesis, Dept. of Electrical Engineering, University of Southern California, August 1987.

[111] INMOS Ltd. Ims 424 transputer - advanced information.

[112] D. G. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley Publishing Co., 1984.

[113] H. Mada. Architecture for optical computing using holographic associative memories. *Applied Optics*, 24, 1985.

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

[115] D. Marr and T. Poggio. Cooperative computation of stereo disparity. *Science*, 194:283–287, 1976.

[116] J. T. Marti. On the convergence of the discrete art algorithm for the reconstruction of digital pictures from their projections. *Computing, Springer-Verlag Inc.*, 21:105–111, 1979.

[117] J. L. McClelland and D. E. Rumelhart. Distributed memory and the representation of general and specific information. *Journal of Experimental Psychology: General*, 114:158–188, 1985.

[118] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in the nervous activity. *Bull. Math. Biophys.*, 5(115), 1943.

[119] C. Mead and L. Conway. *Introduction to VLSI Systems.* Addison-Wesley, 1980.

[120] M. Minsky. *The Society of Mind.* Simon and Schuster, N.Y., 1988.

[121] M. Minsky and S. Papert. *Perceptrons: an Introduction to Computational Geometry.* MIT Press, Cambridge, Massachusetts, 1969.

[122] R. Nevatia. *Machine Perception.* Prentice-Hall, Inc., 1982.

[123] N. J. Nilsson. *Learning Machines.* McGraw-Hill Book Company, 1965.

[124] I. Offen and R.J. Raymond Jr. *VLSI Image Processing.* McGraw-Hill, 1985.

[125] D. Parker. Learning logic. Technical Report TR-47, Center for Computational Research in Economics and Management Science, MIT, Cambridge, 1985.

[126] D. B. Parker. Second order back propagation: implementation an optimal O(n) approximation to Newton's method as an artificial neural network. In *Proc. IEEE Conf. on Neural Information Processing Systems – Natural and Synthetic, Denver*, November 1987.

[127] Behzad K. Parsi and Behrooz K. Parsi. Evaluation of network architectures on test learning tasks. In *Proc. IEEE 1st Intl' Conf. on Neural Networks, San Diego*, pages III 785- III 790, June 1987.

[128] T. Pavlidis. *Structural Pattern Recognition*. Springer-Verlag, 1977.

[129] J. Platt and A. Barr. Constrained network representation. In *Proc. IEEE Conf. on Neural Information Processing Systems – Natural and Synthetic, Denver*, November 1987.

[130] D. C. Plaut and G. E. Hinton. Learning sets of filters using back-propagation. Computer Speech and Languages, 1987.

[131] T. Poggio and C. Koch. Ill-posed problems in early vision: from computational theory to analogue networks. *Proc. R. Soc. London*, 303–323, 1985.

[132] D. Psaltis, A. Sideris, and A. Yamamura. A hierarchical model for voluntary movement and its application to robotics. In *Proc. IEEE 1st Intl' Conf. on Neural Networks, San Diego*, pages IV 551- IV 558, June 1987.

[133] D. Psaltis, K. Wagner, and D. Brady. Learning in optical neural computers. In *Proc. IEEE 1st Intl' Conf. on Neural Networks, San Diego*, June 1987.

[134] L. R. Rabiner and B. H. Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, 3(1):4–16, January 1986.

[135] R. Rastogi, P. K. Gupta, and R. Kumaresen. Array signal processing with interconnected neuron-like elements. In *Proc. IEEE ICASSP'87, Dallas*, pages 54.8.1–54.8.4, 1987.

[136] J. B. Rosen. The gradient projection method for nonlinear programming. Part I, linear constraints. *J. Soc. Indust. Appl. Math.*, 8:181–217, March 1960.

[137] J. B. Rosen. The gradient projection method for nonlinear programming. Part II, nonlinear constraints. *J. Soc. Indust. Appl. Math.*, 9:514–532, December 1961.

[138] F. Rosenblatt. The perception: a probablistic model for information storage and organization in the brain. *Psych. Rev.*, Vol. 65, 1958.

[139] 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)*, Chapter 8, Editors, D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, pages 318–362, MIT Press, Cambridge, Massachusetts, 1986.

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

[141] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group. *Parallel Distributed Processing (PDP): Psychological and Biological Models (Vol. 2).* MIT Press, Cambridge, Massachusetts, 1986.

[142] Editor S. Grossberg. *Neural Networks and Neural Intelligence.* MIT Press, Cambridge, MA, 1988.

[143] T. J. Sejnowski. Parallel networks that learn to pronounce English text. *Complex Syst.*, 1:145–168, 1987.

[144] T. J. Sejnowski and C. R. Rossenberg. A parallel network that learns to read aloud. Technical Report, JHU/EECS-86/01, Johns Hopkins Univ., Dept. of E.E. and C.S., 1986.

[145] K. G. Shin and S. B. Malin. A structured framework for the control of industrial manipulator. *IEEE Trans. on Systems, Man, and Cybernetics*, 15:78–94, January/February 1985.

[146] M. A. Sivilotti, M. A. Mahowald, and C. A. Mead. Real-time visual computations using analog CMOS processing arrays. In *Advanced Research on VLSI*, Editor, P. Losleben, pages 295–312, MIT Press, 1987.

[147] Branko Soucek and Marina Soucek. *Neural and Massively Parallel Computers – The Sixth Generation.* Wiley Inter-Science Publication, John Wiley & Sons, 1988.

[148] F. F. Soulie, P. Gallinari, Y. Le Cun, and S. Thiria. Evaluation of network architectures on test learning tasks. In *Proc. IEEE 1st Intl' Conf. on Neural Networks, San Diego*, pages II 653- II 660, June 1987.

[149] K. Steinbush. The learning matrix. *Kybernetik (Biological Cybernetics)*, 36–45, 1961.

[150] K. Steinbush and B. Widrow. A critical comparison of two kinds of adaptive classification networks. *IEEE Trans. on Electronic Computer*, 737–740, October 1965.

[151] C. V. Stewart and C. R. Dyer. Neural network representation of sensor graphs for autonomous robot navigation. In *Proc. IEEE 1st Intl' Conf. on Neural Networks, San Diego*, pages IV 215– IV 224, June 1987.

[152] G. Z. Sun, H. H. Chen, and Y. C. Lee. Neural network representation of sensor graphs for autonomous robot navigation. In *Proc. IEEE 1st Intl' Conf. on Neural Networks, San Diego*, pages IV 345– IV 356, June 1987.

[153] H. Szu. Fast simulated annealing with Cauchy probability densities. In *Proc. of 2nd Annual Conf. on Neural Nteworks for Computing*, 1986.

[154] M. Takeda and J.W. Goodman. Neural networks for computation: number representations and programming complexity. *Applied Optics*, 25:3033–3046, September 1986.

[155] K. Tanabe. Projection method for solving a singular system of linear equations and its applications. *Numer. Math.*, 17:203–214, 1971.

[156] 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. on Circuits and Systems*, 33:533–541, 1986.

[157] W. K. Taylor. Cortico-thalamic organization and memory. *Proc. Roy. Soc. Lond. Series B*, 159, 1964.

[158] W. K. Taylor. Electrical simulation of some nervous system functional activities. *Information Theory*, 3, 1956.

[159] G. Tesauro and R. Janssens. Scaling relationship in back-propagation learning: dependence on predicate order. Technical report, Center for Complex Systems Research, University of Illinois at Urbana-Champaign, Feburary 1988.

[160] A. P. Thakoor. Content-addressable, high density memories based on neural network models. Technical Report, Jet Propulsion Laboratory JPL D-4166, March 1987.

[161] C. E. Thorp. Path relaxation: path planning for a mobile robot. In *Proc. of the AAAI, Austin*, pages 318–321, August 1984.

[162] S. Tomboulian. Introduction to a system for implementing neural net connections on simd architectures. Technical Report, NASA Langley Research Center, Hampton, NASA Contractor Report 181612, January 1988.

[163] K. Tsutsumi and H. Matsumoto. Neural network representation of sensor graphs for autonomous robot navigation. In *Proc. IEEE 1st Intl' Conf. on Neural Networks, San Diego*, pages IV 525– IV 534, June 1987.

[164] R. Udo and E. Deprettere. On the design of pipelined architecture for the CORDIC algorithm. Technical Report, Delft, University of Technology, The Netherlands, 1985.

[165] J. A. Vlontzos and S. Y. Kung. A hierarchical system for character recognition with stochastic knowledge representation. In *IEEE Int'l Conf. on Neural Networks, San Diego*, pages I 601– I 608, July 1988.

[166] J. E. Volder. The CORDIC trigonometric computing technique. *IRE Transactions on Electron. Comput.*, EC-8(3):330–334, September 1959.

[167] K. Wagner and D. Psaltis. Multilayer optical learning networks. *Applied Optics*, 26:5061–5076, December 1987.

[168] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang. Phoneme recognition using time-delay neural networks. ATR Technical Report, TR-1-0006, ATR Interpreting Telephony Research Lab., 1987.

[169] P. J. Werbos. Beyond regression: new tools for prediction and analysis in the behavior science. Ph.D. Thesis, Harvard University, Cambridge, 1974.

[170] B. Widrow and S. D. Stearns. *Adaptive Signal Processing.* Prentice Hall, Inc., N.J., 1985.

[171] B. Widrow and R. Winter. Neural nets for adaptive filtering and adaptive pattern recognition. *IEEE Computer Magazine*, 21:25–39, March 1988.

[172] G. Widrow and M. E. Hoff. Adaptive switching circuit. *IRE Western Electronic Show and Convention: Convention Record*, 96–104, 1960.

[173] D. J. Willshaw. Models of distributed associative memory models. Ph.D. Thesis, University of Edinburgh, 1971.

[174] D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins. Non-holographic associative memory. *Nature*, 22, June 1969.

[175] Y. T. Zhou and R. Chellappa. Stereo matching using a neural network. In *Proc. IEEE ICASSP'88, New York*, pages 940–943, April 1988.