NEURAL NETWORKS FOR DYNAMIC PROGRAMMING

By

Chinchuan Chiu

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Electrical Engineering

1991

ABSTRACT

NEURAL NETWORKS FOR DYNAMIC PROGRAMMING

By

Chinchuan Chiu

This dissertation presents a fundamentally new and different artificial neural network approach to dynamic programming, including network formulation, analysis, simulation, implementation, and applications. The proposed artificial neural network method is attractive due to its robustness and radically improved speed over conventional techniques. The network, based on the Hopfield model, is defined by an energy function in which the optimal solution corresponds to the lowest energy state. The functionality and the equilibria of the formulated network are analytically proved from the energy function point of view. The quality of the solutions, the network behavior, and the basins of attractions are thoroughly investigated by simulation. An architecture design well-suited to current VLSI technology, in which the network is constructed from neuron array and weight assignment chips, is described. Furthermore, this new approach has been successfully applied to different applications such as optimal control, autoasssociative memory, and speech recognition.

ACKNOWLEDGEMENTS

I sincerely thank Dr. M. A. Shanblatt, my major advisor, for his helpful guidance and support throughout the years of my doctoral studies.

I also thank Dr. E. Goodman, Dr. P. D. Fisher, Dr. R. Enbody, and Dr. C. Ganser for their valuable comments and suggestions on this work. Additionally, I wish to express my gratitude to Dr. J. R. Deller and Mr. M. S. Liu for providing the cepstral coefficients of the vocabulary set used in the speech recognition experiments.

Finally, I want to dedicate this dissertation to my wife Lingming, and my adorable daughters, Honyin and Wileen for they have been sharing love and joy with me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

1.0 Introduction

Artificial neural networks are a fundamentally new and different approach to information processing. They have proven to be very effective in various applications such as pattern recognition, associative memory, control, robotics, and combinatorial optimization. This dissertation presents a new method for dynamic programming using a feedback Hopfield-Tank type network, including network formulation, theory, implementation, and applications.

This Chapter begins with a brief overview of the field of neural networks. The problem to be solved in this dissertation is then stated, followed by the research tasks. Finally, the organization of this dissertation is outlined.

1.1 Overview of Neural Networks

The past few years have seen a resurgence of research in the field of neural networks promoted by advances in electronic circuit technology as well as a deeper understanding of the functioning of brain[1-3]. One motivation is a desire to build a new breed of powerful computers (the sixth generation) to solve a variety of problems that are very difficult to solve, or at least computationally cumbersome, with conventional digital

computers. Cognitive tasks such as recognizing a familiar face, learning to speak, and understand a natural language, retrieving contextually appropriate information from memory, and guiding a mechanical hand to grasp objects of different shapes and sizes are representative examples of these problems [4-6]. All require some form of pattern recognition, pattern matching, nonlinear discrimination, and/or combinatorial optimization. Quite interestingly, these tasks are analogous to those typically performed naturally and robustly by the brain. For example, humans can usually recognize a familiar face in about 200 milliseconds [7]. The state-of-the-art image processing system can not even come close to this performance. Moreover, the brain is remarkable in that this performance is obtained by a system whose individual components, neurons, are individually much slower than currently used electronic components [7,8].

Another motivation is a desire to develop cognitive models that can serve as a foundation for artificial intelligence [9-11]. Although it is well known that the brain is not as good as a digital computer at performing arithmetic operations, there are several aspects of brain function that we are not able to duplicate with conventional computers. Some of these are association, categorization, generalization, feature extraction, and optimization [12,13].

Many researchers have been developing models to study neural networks. These models fall into two categories. One is biological modeling with a goal to study the structure and functions of the real brain in order to explain biological behaviors. The other is technological modeling which studies brains in order to extract concepts to be utilized in new and more powerful computer architectures [14-16]. Although there is a controversy over which one of these two branches should constitute the true focus of research in neural network modeling, the latter path has been taken by many working in the area of what has come to be called artificial neural networks (ANN).

An ANN is a parallel distributed information processing system which consists of processing elements (neurons) and connections (synapses). Each processing element,

characterized by its working function, can receive input signals from a certain number of incoming connections and produce output signals to a certain number of outgoing connections. Table 1-1 contains a comparison of some of the characteristics of neural networks and conventional digital computers. Stored information and network output of ANN is determined by its structure, processing elements, and connections. Since the output results are collective decisions of all parameters together, the loss of information due to faults such as damaged connections and malfunctioning processing elements will still result in correct outputs as long as other parts of the network overrule the faulty parts. This provides the inherent fault tolerance property for ANNs. The ability of massive parallel computation due to many processing elements operating in parallel is another important feature possessed by ANNs, which is essential for the applications requiring high computation rate such as pattern recognition, and combinatorial optimization.

The history of neural network research has been intertwined with symbolic artificial intelligence since the beginning of both fields [17-19]. In the late 1950s, when a group of scientists turned their attention to attempting to build intelligent systems, two different approaches emerged. One focused on how the brain did things, the other concentrated on what the brain did regardless of how it was accomplished biologically. The second approach, artificial intelligence, became favored over the first approach because of rapid advances in software technology which provided flexible and powerful tools for testing models and concepts. In the early 1980s, several key papers published by Hopfield, Grossberg, and Rumelhart spurred a revival of interest in the first approach - neural networks [2-4].

The Hopfield-Tank network, one of the most well-known artificial neural networks, has been applied to combinatorial optimization problems such as the traveling salesman problem (TSP) [20-23], linear programming problems [24,25], nonlinear programming problems [26,27], communication network routing problems [28,29], and others [30-33]. Due to the gradient property of the Hopfield-Tank network [2], each of these problems can

Characteristics	Neural Networks	Conventional Computers
Memory Structure	Distributed	System-Dependent
Memory Recall	Associative	Specific Input
Fault-Tolerance	Inherent	Not Inherent
Pattern Recognition	Excellent	Poor
Classification	Excellent	Poor
Learning	Excellent	Poor
Arithmetic Operation	Poor	Excellent
Timing Scheme	Asynchronous	System-Dependent
Degree of Parallelism	High	System-Dependent
Processing Element	Simple	Complex
Degree of Connectivity	High	Low

Table 1-1. Characteristics of neural networks and conventional computers.

be solved by designing an appropriate network whose minimum energy states correspond to the problem's solutions.

In particular, a dynamic programming neural network has been developed for dynamic programming problems using a feedback Hopfield-Tank type network [34,35]. This network formulation has been shown to robustly provide near-optimal solutions in a very short period of time independent of network size. It has also been shown to be a gradient system based on the fact that the derivative of the associated energy function along all trajectories is less than or equal to zero. The network can converge to one of the stable equilibria (minimum states of the associated energy function) if the initial conditions are sufficiently close to the stable equilibrium. The stable equilibria can be regarded as the near-optimal solutions to the problem.

1.2 Statement of Problem

A typical dynamic programming problem (DPP), shown in Figure 1-1, can be modeled as a set of source and destination nodes with n intermediate stages, m states in each stage, and metric data $d_{xi,\,(x+1)j}$, where x is the index of stages, and i and j are the indices of the states in each stage. The object is to find an optimal path composed of one and only one state in each stage from source to destination.

The conventional approach uses the *principle of optimality* which requires a large number of computations to determine the optimal solution [35]. In fact, the number of computations is so great for large dynamic programming problems, that the conventional algorithms are not effective in many applications. This is especially true for those requiring fast, perhaps real-time, solutions. Additionally, computation in the conventional algorithms for dynamic programming is executed often sequentially, therefore the computation time is proportional to the problem size. This is primarily due to the fact that the basic algorithms are only partially parallelizable.

Interestingly though, in many real-time dynamic programming applications, the

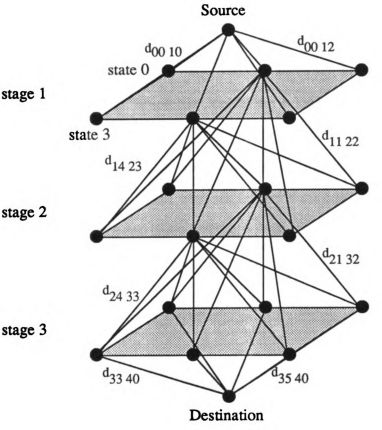


Figure 1-1. A 3x6 dynamic programming problem (full interconnection not shown).

rapid calculation of near-optimal solutions is sufficient when contrasted to a slowly computed globally optimal solution. For example, robot trajectory planning problems, aircraft trajectory control problems, and other optimal control problems that must respond quickly to radically changing environmental conditions are of this type.

To solve dynamic programming problems with an artificial neural network, an appropriate network formulation must first be determined such that the problem solution corresponds to the global optimal solution, or at least a local near-optimal solution. The essential issues of stability and convergence of the formulated network must be ensured so that the network always converges to one of the stable equilibria. In order to obtain good solutions for problems, the qualitative network behavior must be understood. As will be seen in Chapter 3, this network characteristic is determined by its associated energy function. Finally, some of the implementation problems for this network and networks of this type must be addressed.

1.3 Research Tasks

The tasks of this research are to (1) formulate an artificial neural network for dynamic programming; (2) analyze the formulated network from the view point of associated energy function; (3) verify the formulation and analysis through network simulation; (4) demonstrate the applicability of the formulated network by solving optimal control, associative memory, and speech recognition problems; and (5) propose an architecture for the formulated network.

The network formulation will be based on the Hopfield-type network. To design an Hopfield-type network for solving a specific problem and determine the connection weights and parameters to achieve the best solutions to the problem, a general procedure goes as follows:

(1) Select an encoding procedure such that the outputs of the network correspond to the solutions of the problem.

(2) Choose a proper energy function, bounded from below, whose minimum corresponds to the optimal solution of the problem.

(3) Determine connection weights and bias current which properly represent the objective function and constraints of the problem.

To formulate a neural network for dynamic programming, each state node will be considered as an individual neuron. Examining the characteristics of the optimal path carefully, two constraints become evident. First, the optimal path must visit one and only one state in each stage (a structural constraint). Secondly, the optimal solution must have the minimum total cost based on the given performance measure (a cost constraint). Thus, the energy function has two requirements. The structural constraint implies that the energy function must converge to stable states where one and only one state in each stage is active. The cost constraint dictates that the energy function must converge to stable states representing a minimum path. After the associated energy function has been determined, the network can be formulated as will be shown in Chapter 3.

The second task is a two-fold analysis of the network. The first objective is to explain why the formulated network is functional by investigating the following properties.

(1) The network must possess a unique solution for every initial condition.

(2) The time derivative of the associated energy function decreases monotonically along the trajectories and becomes zero only at equilibrium points.

(3) The network must have a finite number of equilibrium points.

(4) Every equilibrium point of the network must be a local minimum of the associated energy function.

Property 1 will ensure the existence and uniqueness of the solution. Property 2 will guarantee the stability and convergence of the network. The possibility of existing periodic solutions will be avoided by Property 3. Finally, Property 4 will provide a way to find the equilibria from the associated energy function. Once the above properties are verified, the network will be shown to converge to one of the equilibria, if the initial condition is in one

of the basins of attraction of the equilibrium points.

The goal of designing the network is to obtain near-optimal solutions for dynamic programming problems. Therefore, the equilibria must reside in the regions which can be regarded as valid and near-optimal solutions. The second objective then is to analyze the relationship between the quality of the solutions and the parameters of the associated energy function, derivation of the locations and numbers of the minimum states for different components of the energy function, and the locations of the minimum states of the energy function with different values of parameters.

The third task involves a simulation which will demonstrate the functionality of the formulated network and show that it can produce near-optimal solutions for DPPs. Since the network is described by a set of first order differential equations, the network can be simulated by solving the associated differential equations with numerical analysis techniques running on a conventional computer. Simulation programs written in C and based on a fourth-order Runge-Kutta method for the formulated network will be developed.

This task will be divided into four sub-tasks which are to:

(1) depict the relationship between the quality of solutions and the parameters of the associated energy function;

(2) simulate different networks ranging from 2×2 to 64×64;

(3) simulate networks with random errors to connection weights;

(4) simulate networks with different initial conditions and different parameters of the associated energy function; and

(5) simulate large-scale dynamic programming problems with networks of smaller size by decomposing the problems.

The first sub-task can provide appropriate values of the parameters for the subsequent simulations. The results of the second sub-task will serve as an index to the success of the formulation of the networks. The Hopfield-Tank networks are shown to be gradient systems only when the networks are symmetric. However, it is believed that

sufficiently small changes of the connection weights will not alter the qualitative properties of the networks [37]. The third sub-task will show the network sensitivity to errors in the connection weights. The fourth sub-task will qualitatively show how the parameters affect the basins of attraction of the equilibrium points.

The implementation of large scale neural networks has been severely restricted by current technology. One way to cope with this problem is to use a divide-and-conquer method. The advantage of this method is that the size of the network used to solve larger problems will remain small. However, the time to find a solution will increase accordingly. In the last sub-task, the reduction of the network connectivity, the problem-solving time, and the quality of the solutions will be examined.

For the fourth task, three problems in different applications, optimal control, associative memory, and speech recognition, will be solved by the formulated network. Dynamic programming is one of the methods which have been used to solve optimal control problems. A continuous system described by a first order differential equation will be used as an example to show how dynamic programming neural networks can produce near-optimal solutions based on a predefined performance measure. Before the network can be applied, the system differential equation must be approximated by a corresponding difference equation, and the performance measure must be expressed or approximated in a discrete form. Then the structure and connection weights of the network can be determined. The solutions produced by the networks will be compared to the optimal solutions.

One of the most useful and most explored areas of applications for artificial neural networks is associative memory. Suppose that a set of memory patterns $\{Y^1, Y^2, ..., Y^l\}$ is to be stored, where $Y^r \in R^n$ for $r = 1$ to l. Given a stimulus $X = Y^r + \varepsilon$ where ε is sufficiently small, an associative memory is designed in such a way that the output pattern Y^r can be recalled successfully. The Hopfield-Tank model is reportedly capable of implementing associative networks [7]. However, some of the proposed networks do not guarantee that

each desired memory pattern can be stored [7,38,39], and some of them suffer from a rather complex synthesis procedure [40,41]. The network can be used to store patterns by selecting an appropriate working function for processing elements and connection weights between processing elements so that the patterns correspond to the minima of the network.

In the template matching approach for speech recognition, the time scales of a test and a reference speech pattern are generally not perfectly aligned, therefore a nonlinear time warping is required to compensate for local compression and/or expansion of the time scales in order to find the best difference measure between them. Dynamic time warping methods based on dynamic programming techniques have been developed to perform time alignment and evaluate the difference between a test and a reference speech pattern [42-45]. The network for dynamic time warping is designed in such a way that the minimum states of the network's energy function correspond to the near-best correlation between a test and a reference pattern [46].

Although artificial neural networks are composed of basic circuits emulating neurons and synapses, the implementation of ANNs has been severely restricted in size by current VLSI technology limits such as real-estate considerations, and fan-in, fan-out, and connectivity limitations [47, 48]. To alleviate some of these implementation problems, a building block paradigm, in which neural networks are constructed by numbers of VLSI chips implementing neurons and synapses separately, will be used. Eberhardt, *et. al.* have designed various CMOS VLSI building block components in a single chip, such as a synapse chip and neuron array chip, for feed-forward networks [47]. Based on their designs, a simple, regular, and implementable architecture will be proposed for dynamic programming neural networks. In addition, the proposed architecture can serve as a co-processor such that the connection weights can be programmed via a host computer so that different DPPs may be solved.

1.4 Organization of This Dissertation

This dissertation is organized as follows. Chapter 2 contains a background discussion of the appropriate topics. Chapter 3 presents a general neural network structure for dynamic programming and analyzes the network formulation. For the first part, an artificial neural network formulation for solving the dynamic programming problem is presented. The formulation procedure is implemented and demonstrated using a modified Hopfield-Tank ANN. In the second part, an analytical examination of the energy function associated with a dynamic programming neural network is presented. The analysis is carried out in two steps. First, the locations and numbers of the minimum states for different components of the energy function are investigated in the extreme cases. Then, the locations of the minimum states of the energy function using different parameter values are derived. This analysis provides a theoretical foundation for dynamic programming neural networks.

Chapter 4 presents simulation results which will demonstrate the functionality of the formulated network and show that it can produce near-optimal solutions for different problems. The relationship between the quality of solutions and the parameters of the associated energy function, the network behavior due to random errors in connection weights, and the basins of attraction of the network's equilibria are studied. Moreover, the simulation results of different networks ranging from 2×2 to 64×64; are also presented.

Chapter 5 proposes an building block architecture for dynamic programming neural networks and a problem decomposition method. The proposed architecture can serve as a co-processor such that the connection weights can be programmed via a host computer so that different DPPs may be solved. The problem decomposition based on a divide-and-conquer method can be used to solve large-size dynamic programming problems with networks of smaller size.

Chapter 6 presents a new autoassociative memory network using dynamic programming neural networks.The formulation, based on dynamic programming techniques used to find minimum cost paths, is very simple and straightforward in the sense that memory

patterns can be easily added to an existing associative memory without any modification to the previous network connections. Two examples, with the first one designed to correct single-error codes and the second to distinguish three images, are used to demonstrate the power of the formulated network. The issue of fault tolerance on the autoassociative memory network is also considered. Comparisons between the proposed network and the other two networks are also given in this chapter.

Chapter 7 presents a neural network method for dynamic time warping on speech recognition. The network for dynamic time warping is designed in such a way that the minimum states of the network's energy function correspond to the near-best correlation between a test and a reference pattern. To achieve real-time applications, an architecture based on this method is also proposed.

Finally, Chapter 8 describes the contributions and conclusions of this dissertation.

Chapter 2

2.0 Background

This chapter begins with an introduction of artificial neural network models. The Hopfield model is then presented. Finally, the dynamic programming problems which will be dealt with in this dissertation are defined.

2.1 Artificial Neural Network Models

Scientists and researchers have been trying to build neural network models for the purpose of studying the function of the brain for many years. These models, generally involving very complicated mechanisms and structures, fall into two categories. In biological modeling, which is of primary interest to neurobiologists, the goal is to study the structure and function of the real brains in order to explain biological behaviors. In technological modeling, which is of primary interest in this work, the goal is to study brains in order to extract concepts for the new generation computer architectures. Although there is a controversy over which one of these two branches should constitute the true focus of research in neural network modeling, the latter path has been taken by many working in the area of artificial neural networks.

An artificial neural network is a parallel distributed information processing system

consisting of processing elements (neurons) and connections (synapses). Each processing element, characterized by its working function, can receive input signals from a certain number of incoming connections and produce output signals to a certain number of outgoing connections [49].

Neural networks incorporate a combination of certain features of various information processing systems together with some special features. These special features include the use of simple processing elements, learning abilities to adjust parameters and connection weights to give the desired responses as well as to compensate for inaccuracies and faults in the hardware, and fine-grain and massive-scale parallel problem solving ability. The combination of these features results in significant advantages for several classes of information processing problems. Massively parallel algorithms can provide rapid processing if implemented on an appropriate processing architecture, especially one which is matched to the algorithm. The use of only a few different type of simple processing elements can facilitate fabrication of such massively parallel systems. The adaptive feature of the processing elements also allows some element inaccuracies to be compensated. Learning ability also provides a very important advantage in dealing with the detailed knowledge necessary to build the system.

A neuron is the basic cell of a neural network. The simplest artificial neuron model was introduced by McCulloch and Pitts [50]. This model sums the weighted inputs and passes the result through a certain function. The output of the function, considered as the output of the neuron, is branched out via weighted connection to the inputs of other neurons. This simplified representation is shown in Figure 2-1 where x_i's are the inputs from other neurons, w_{ij} is the weight of the connection from the output of neuron i to the input of neuron j, $g(\cdot)$ is the neuron working function, θ_j is the threshold value, and y_j is the output.

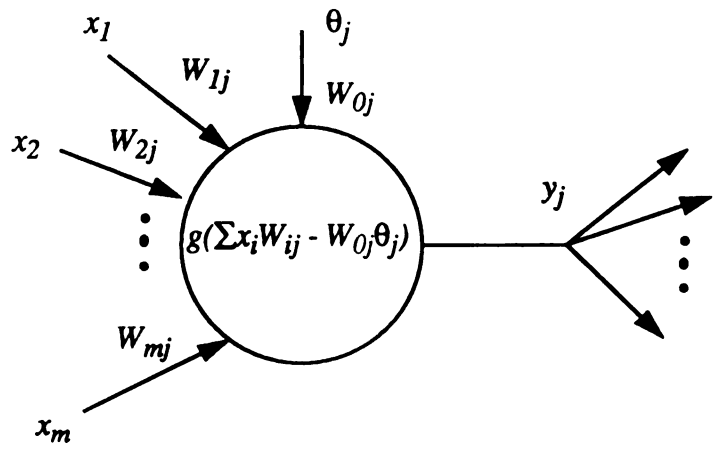
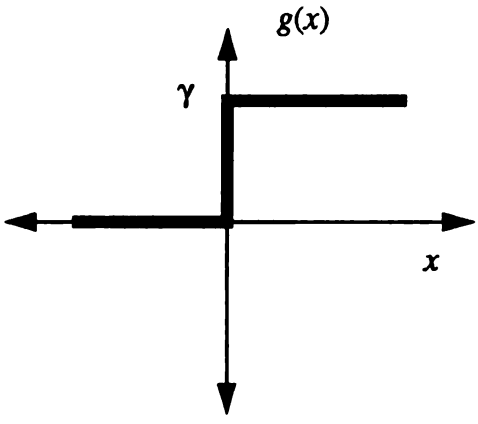


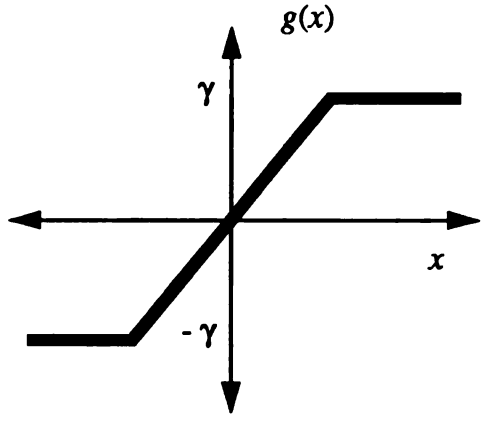
Figure 2-1. Simplified representation of a neuron.

The neuron working function is commonly one of the three types: hard limiters, threshold logic elements, and sigmoid nonlinearities. These input/output relationships are shown in Figure 2-2. A more complex neuron may include temporal integration or other types of time dependencies and more complex mathematical operations than summation [51]. The sigmoid nonlinearity is the most commonly used because it is bounded, monotonically increasing, and most resembles a real neuron working function. The range of the sigmoid function can be [0, 1] for binary mode or [-1, 1] for bipolar mode, depending on the application.

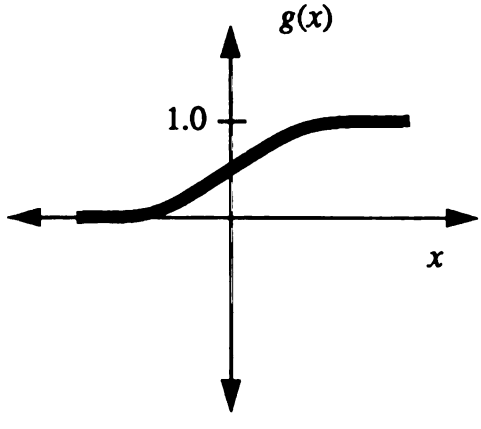
Groups of artificial neurons can be interconnected in a variety of ways to form ANNs. Many topological configurations for ANNs have been proposed. These networks fall into two categories: feed-forward and feedback networks. A feed-forward network is shown in Figure 2-3. The signal flow of a feed-forward network is unidirectional. Feed-forward networks can be single-layered or multi-layered. Figure 2-4 shows a feedback network. Both forward and backward connections exist in feedback networks. Feed-forward networks have been used in applications such as pattern recognition [52-56],



(a) Hard limiter.



(b) Threshold logic element.



(c) Sigmoid nonlinearity.

Figure 2-2. Neuron working functions.

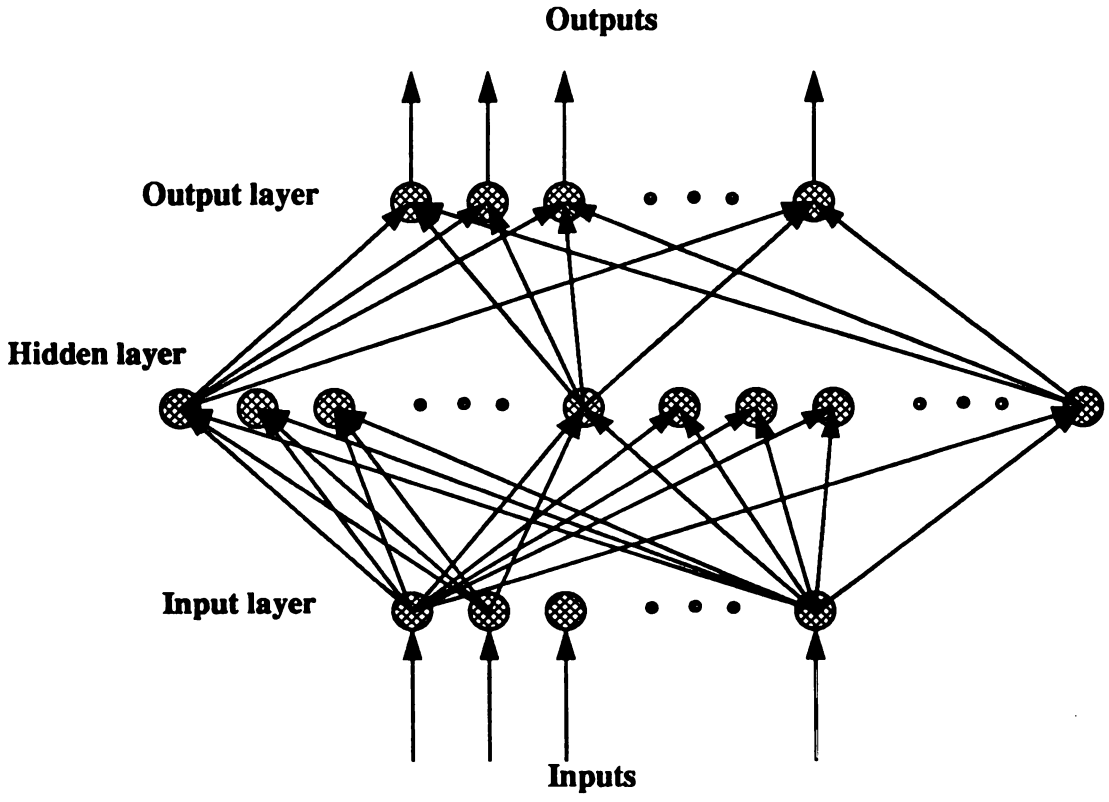


Figure 2-3. A 2-layered feed-forward neural network (connections are not fully shown).

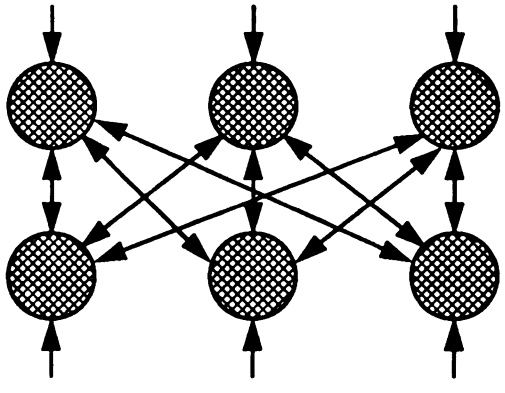


Figure 2-4. A feedback neural network.

robotics [57], and control problems [58-60]. However, feedback networks are useful in optimization [61,62], object recognition [63-65], and associative memory problems [11,12].

2.1.1 Single-layered Perceptrons

A single-layered feed-forward ANN can be created from an array of n artificial neurons. Each of these receives m inputs x_1, x_2,..., x_m, via corresponding weights. This network is able to learn to recognize simple patterns via a convergence procedure for adjusting weights which stores patterns in a collective fashion [66,67]. This convergence procedure goes as follows. First, connection weights and the threshold values, each of which can be considered as another weight associated with a fixed input value, are initialized to random nonzero values. Second, a training pattern with n features is applied to the input and then the output is computed. Third, the connection weights are adapted according to the following equation

$$W_{ij}(t+1) = W_{ij}(t) + \eta\,[d_j(t) - y_j(t)]\,, \tag{2-1}$$

where d_j(t) is the desired output of neuron j, t is the number of iterations, and η is a positive gain factor less than 1. The procedure is repeated until the values of W_{ij}(t+1) - W_{ij}(t) are less than a predefined value for all i and j.

In the simplest case with n = 1, the network is a single-neuron network shown in Figure 2-1. This network, called a perceptron, can decide whether an input belongs to one of two classes (denoted as class A or B) after a sufficient training process. The perceptron uses a hard limiter such that the output is either 1 or -1. The network will respond as class A if the output is 1 and class B if the output is -1.

Unfortunately, the single-layered perceptron is not able to solve problems where classes cannot be separated by a hyperplane or a line in two dimension [19,25]. The XOR

problem which is nonseparable by a line is an example of this problem type.

2.1.2 Multi-layered Perceptrons

Multi-layered perceptrons are feed-forward networks with one or more layers of neurons between the input and output layer. A two-layered perceptron with one hidden layer is shown in Figure 2-3. With the development of the back-propagation training algorithm, multi-layered perceptrons overcome many of the limitations of single-layered perceptrons [3]. The multi-layered perceptrons with a back-propagation training algorithm have been tested successfully with a number of deterministic problems such as the XOR problem [3], pattern recognition problems [11], and others [68,69].

The back-propagation algorithm uses a gradient search technique to minimize a cost function equal to the mean square difference between the desired and the actual outputs [3]. The network is trained by initially selecting the weights randomly and then the intermediate outputs corresponding to a given input pattern from layer to layer in a forward fashion are calculated. The final layer output is compared to the desired output and the error is passed in a backward direction to adjust the connection weights. The modification of weights is carried out from layer to layer in a backward fashion. The weights are adjusted after each trial until weights converge and the cost function is reduced to an acceptable value [3].

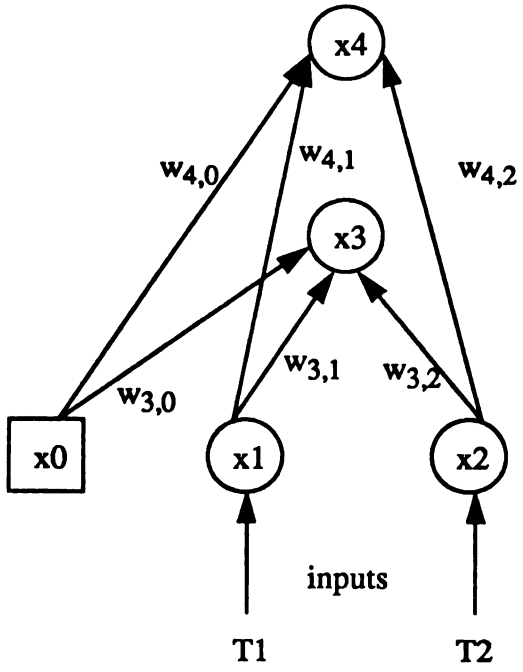
2.1.3 An Example

One of the most famous backpropagation networks is the XOR multi-layered perceptron. This network was used to solve the logic exclusive-or problem by training with examples and can be also used to show how neural networks do information processing. Though not particularly interesting from an application perspective, the exclusive-or problem was not solvable using the neural networks of the early 60's [8].

Figure 2-5 shows this network. Processing elements 1 and 2, which are in the input layer, simply pass whatever input is applied to them directly to their output. Processing

elements 3 in the hidden layer and 4 in the output layer function the same way as shown in Figure 2-1. The internal activation level for processing elements 3 and 4 is computed from the weighted sum of the inputs. This internal activation level is transformed into the output by its sigmoid function. The lines between processing elements are connections. The arrows show the direction of information flow of a connection. The weight for each connection modulates the output value of the source processing element before it is passed on to the destination. The two inputs, T1 and T2, can take on any value between 0.0 and 1.0. The output ranging from 0.0 to 1.0 of processing element 4 is the output of the network.

The training process involves applying T1 and T2 to the network, and presenting the corresponding output to the network. With each iteration, the network slightly changes the strength of the connection weights. Over the course of about 100 iterations of each case in the training set, the weights converge to the results shown in Figure 2-5.



$w_{3,0} = 2.8$, $w_{3,1} = -7.4$, $w_{3,2} = -7.4$

$w_{4,0} = 8.6$, $w_{4,1} = -5.6$, $w_{4,2} = -5.6$

Figure 2-5. An XOR network.

2.2 The Hopfield-Tank Model

The Hopfield-Tank model falls into the feedback network category. One of the major contributions of the Hopfield-Tank model is that it can be built with analog components and is suitable for analog VLSI implementation [70-74]. In this model, each neuron with input u_i and output v_i is modeled as an amplifier with a capacitive element C_i and a resistive element ρ_i at the input node. These components define the time constant of the neuron. The output of neuron j is connected to the input of neuron i via a conductance T_{ij}. Figure 2-6 illustrates the basic neuron structure used in the Hopfield-Tank model. The input-output relationship of the amplifier is sigmoidal.

A general Hopfield-Tank network is shown in Figure 2-7. This network can be described by a set of first-order differential equations of the form [14,15]

$$C_i\left(\frac{du_i}{dt}\right) = \sum_{j=1}^{n} T_{ij}v_j - \frac{u_i}{R_i} + I_i, \tag{2-2.1}$$

where

$$\frac{1}{R_i} = \frac{1}{\rho_i} + \sum_{j=1}^{n} T_{ij}, \tag{2-2.2}$$

$$v_i = g_i(u_i), \tag{2-2.3}$$

n is the number of neurons, I_i is the external input current, and g_i is a sigmoid function.

The energy function selected by Hopfield for this network is

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} T_{ij}v_iv_j - \sum_{i=1}^{n} I_iv_i + \sum_{i=1}^{n}\frac{1}{R_i}\int_0^{v_i} g_i^{-1}(\xi)\,d\xi \tag{2-3.1}$$

such that

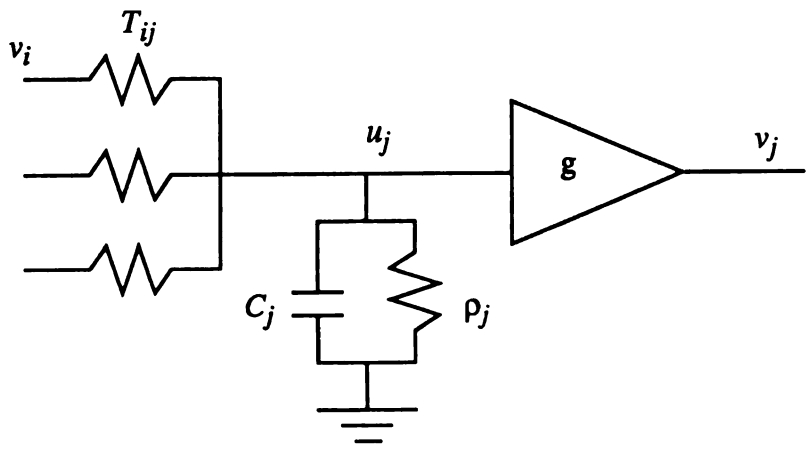


Figure 2-6. A basic neuron of the Hopfield-Tank model.

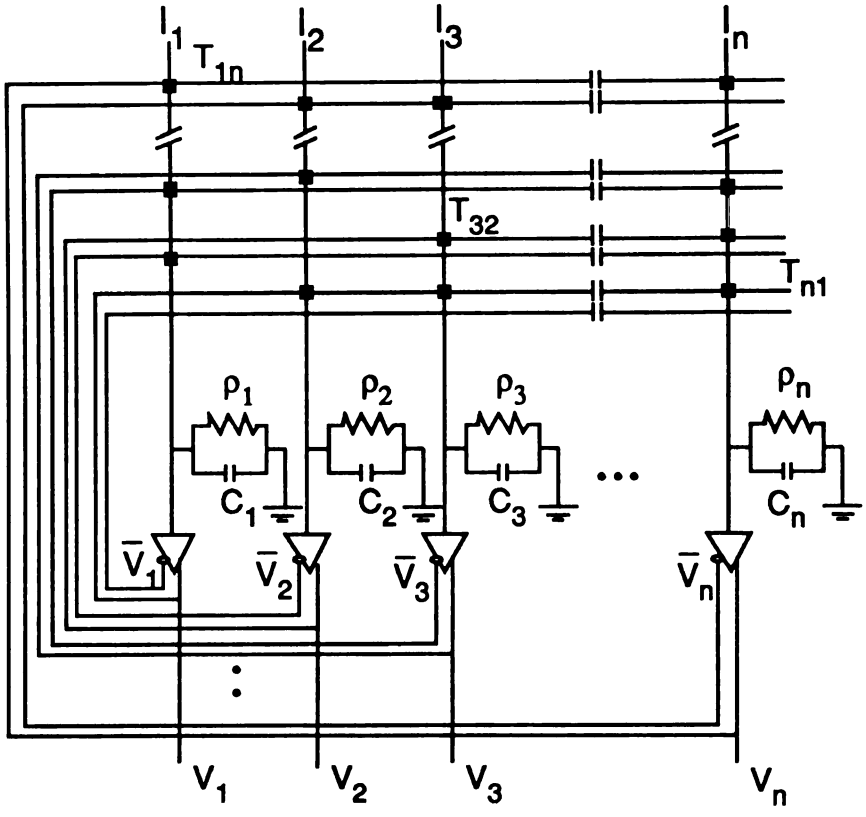


Figure 2-7. A general Hopfield-Tank artificial neural network.

$$C_i\left(\frac{du_i}{dt}\right) = -\frac{\partial E}{\partial v_i}. \tag{2-3.2}$$

Suppose $T_{ij} = T_{ji}$ for all i and j, then the time derivative of the energy function along the trajectories can be derived by applying the chain rule as

$$\frac{dE}{dt} = \sum_{i=1}^{n} \frac{\partial E}{\partial v_i}\frac{dv_i}{du_i}\frac{du_i}{dt} = -\sum_{i=1}^{n} \frac{1}{C_i}\frac{dg_i(u_i)}{du_i}\left(\frac{\partial E}{\partial v_i}\right)^2. \tag{2-4}$$

Since $g_i(u_i)$ is monotonically increasing, $\frac{dE}{dt} \le 0$ for all t. As a result, the value of the energy function is strictly decreasing and becomes zero only at equilibrium points where $\frac{dE}{dt} = -C_i\frac{du_i}{dt} = 0$ for all i.

If u_i is replaced by λu_i in equation (2-2.3), where λ is a constant representing the neuron gain, then we have

$$v_i = g_i(\lambda u_i), \tag{2-5.1}$$

$$u_i = \frac{1}{\lambda} g_i^{-1}(v_i), \tag{2-5.2}$$

and

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} T_{ij}v_i v_j - \sum_{i=1}^{n} I_i v_i + \sum_{i=1}^{n} \frac{1}{\lambda R_i}\int_0^{v_i} g_i^{-1}(\xi)\, d\xi. \tag{2-5.3}$$

If λ is chosen to be very large, then the integral term of the energy function is negligible compared to the other terms. This leads to the following:

$$C_i\left(\frac{du_i}{dt}\right) = \sum_{j=1}^{N} T_{ij}v_j + I_i, \tag{2-6.1}$$

and

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} T_{ij}v_i v_j - \sum_{i=1}^{n} I_i v_i \cdot \tag{2-6.2}$$

These two equations are valid only for high gain limit, that is, when λ is very large.

Equations (2-2.1) - (2-4) actually define a gradient system and thus guarantee no oscillation or any complicated behavior in the system [14,75,76]. Furthermore, it has been proven that such a system has only a finite number of equilibria if the equilibria are isolated [40,41]. Those isolated equilibria may correspond to memory patterns in associative memory, feature patterns in pattern recognition problems, or solutions to an optimization problem.

2.3 Dynamic Programming Problems

Dynamic programming is a very useful mathematical technique for making a sequence of interrelated decisions. It provides a systematic procedure for determining the combination of decisions that maximizes or minimizes overall performance measures. However, there does not exist a standard formulation for dynamic programming problems. Rather, dynamic programming is a general approach to problem solving, and the particular equations must be developed to fit each individual situation. The following example shows how it can be solved by traditional dynamic programming procedures. This dynamic programming problem has been shown in Figure 1-1. A performance measure is defined as the total length of a valid path from the source node to the destination node. Given the source and destination nodes, the number of stages n, the number of states in each stage m, and the metric data $d_{xi,\,(x+1)j}$, where x is the index of stages, and i and j are the indices of

states in each stage, the problem is to find an optimal path from source to destination. This optimal path is measured with respect to a performance criterion.

To illustrate the conventional algorithm for solving the DPP, the principle of optimality is presented first. Suppose the optimal path for a multistage decision problem starting at w and reaching z consists of segment w-x with cost C_{wx} and segment x-z with cost C_{xz}. The minimum cost C* is equal to the sum of C_{wx} and C_{xz}. The principle of optimality states that if w-x-z is the optimal path from w to z, then x-z is the optimal path from x to z. This principle can be proven by contradiction. Suppose there exists an optimal path x-y-z which is different from x-z. Then C_{xyz} must be less than C_{xz} and

$$C_{wx} + C_{xyz} < C_{wx} + C_{xz} = C^*.$$

Since C* is the minimum cost from w to z, the principle of optimality is proved.

Traditional dynamic programming is a computational technique which makes a sequence of decisions to define an optimal path based on the principle of optimality. The conventional algorithm begins by finding the optimal path for the last stage and moves backward stage by stage until the optimal path starting at the source node is found. This procedure can be stated with the help of Figure 1-1 as follows.

At the final stage (i.e, stage 3),

$$L^*_{3i\,40} = d_{3i\,40}$$

= the minimum length from state *i* of stage 3 to the destination.

At stages k=2 and k=1,

$$L^*_{ki\,40} = min\,\{d_{ki\,(k+1)\,j} + L^*_{(k+1)j\,40}\}$$

= the minimum length from state *i* of stage k to the destination.

At the source node

$$L^*_{00\,40} = min\,\{d_{00\,1j} + L^*_{1j\,40}\}$$

= the minimum length from the source to the destination,

where $i = 0,1...,5$ and $j = 1,.,3$.

Since the conventional algorithm finds the optimal path in a sequentially backward stage-by-stage fashion, the algorithm cannot be fully parallelized. In addition, due to the limitations of computation capacity, synchronization, communication, and data distribution problems, high-speed digital computers with some degrees of parallelism may have difficulties in dealing with large-scale problems.

The previous problem is a typical example of dynamic programming problems. One way to recognize a situation that can be formulated as a dynamic programming problem is to notice its basic structure is analogous to that of the previous problem. For this dissertation, the dynamic programming problems which are considered solvable by the proposed neural network method have the following basic features.

(1) The problems can be divided into stages, with a deterministic policy decision required at each stage.

(2) Each stage has a finite number of states associated with it.

(3) The effect of the policy decision at each stage is to transform from the current state into a state associated with next stage.

(4) The solution procedure is designed to find an optimal solution for the problems.

(5) Given the current state, an optimal sub-solution for the remaining stages is independent of the sub-solution adopted in previous stages.

Chapter 3

3.0 Network Formulation and Analysis

Chapter 3 consists of two major parts, network formulation and network analysis. For the first part, an artificial neural network formulation for solving the dynamic programming problem is presented. The problem entails finding an optimal path from a source node to a destination node which minimizes (or maximizes) a performance measure of the problem. The optimization procedure is implemented and demonstrated using a modified Hopfield-Tank network. The proposed network for dynamic programming is attractive due to its radically improved speed over conventional techniques, especially where real-time near-optimal solutions are required.

In the second part, an analytical examination of the network is presented. First of all, propositions related to the functionality of the network are proved. Second, a two-step energy function analysis is carried out in order to explain how the dynamic programming neural network can provide near-optimal solutions, since the network behavior is intimately related to the associated energy function. First, the locations and numbers of the minimum states for different components of the energy function are investigated in the extreme cases. A clearer insight of the energy function can be visualized through the minimum states of different components. Second, the locations of the minimum states of the

energy function using different parameter values are derived. It is shown that the minimum states can reside in regions which are regarded as valid solutions with certain conditions.

3.1 Network Formulation

Generally, the procedure for solving a specific problem using the Hopfield network involves three steps. First, an encoding procedure is selected such that the outputs of the network correspond to the solution(s) of the problem. Second, a proper energy function, whose minimum corresponds to the optimal solution of the problem, is determined. Third, according to the determined energy function, connection weights and bias currents which properly represent the objective function and constraints of the problem are defined

By giving an appropriate working function to each processing element and an associated weight for each connection between two processing elements, a properly configured network can rapidly and robustly provide at least a locally optimal solution due to the nature of the collective computations. Although the topology of the optimization surface in the solution space is so complicated that the globally optimal solution is not guaranteed, many good solutions which are at least locally similar to the optimal solution can be obtained.

Consider again the 3x6 DPP shown in Figure 1-1. The goal is to find a valid path which starts at the source node, visits one and only one state node in each stage, reaches the destination node, and has a minimum total length (cost) among all possible paths. A properly designed network is defined by an energy function in which the optimal solution corresponds to the lowest energy state of the network. Examining the characteristics of the optimal path carefully, two constraints become evident. First, the optimal path must visit one and only one state in each stage (a structural constraint). Secondly, the optimal solution must have the minimum total cost based on the given performance measure (a cost constraint). Thus, the energy function has two requirements. The structural constraint implies that the energy function must converge to stable states where one and only one state in each stage is active. The cost constraint dictates that the energy function must converge

to stable states representing a minimum path.

Each state node can be conveniently considered as an individual processing element. To develop an appropriate energy function for the network, v_{xi} is taken as the output of a processing element of the ith state in the xth stage, where n is the number of intermediate stages. The following formal constraints are thus defined.

To satisfy the structural constraint, one and only one processing element must be active in each stage and the number of active processing elements equal to the number of stages. This constraint is embedded in

$$E1 = \sum_x \sum_i \sum_{j \neq i} v_{xi} v_{xj} + \left(\sum_x \sum_i v_{xi} - n \right)^2 . \tag{3-1}$$

And, the corresponding cost constraint is given by

$$E2 = \sum_x \sum_i \sum_j d_{xi,(x+1)j} v_{xi} v_{(x+1)j} + d_{(x-1)j,xi} v_{xi} v_{(x-1)j} . \tag{3-2}$$

$E1$ will vanish for a valid path. For a minimum cost path, $E2$ takes on a minimum value. Let the sigmoid function $g(u_{xi}) = (1/2)(1 + \tanh(u_{xi} / \lambda))$. From equation (2.3.1), we define the energy function for the dynamic programming network as

$$E = \frac{a}{2} E1 + \frac{b}{4} E2 + \sum_x \sum_i \int_{0.5}^{v_{xi}} \frac{1}{R_{xi}} g_{xi}^{-1}(\xi)\, d(\xi) \tag{3-3}$$

where a and b are positive numbers.

The quadratic terms in equation (3-3) define the connection weight matrix T and the linear term defines the bias current vector I. From the circuit equation of the Hopfield network,

$$C_{xi} \left(\frac{du_{xi}}{dt} \right) = \sum_{yj} T_{xi,yj} v_{yj} - \frac{u_{xi}}{R_{xi}} + I_{xi} . \tag{3-4}$$

the weight of the connection linking the *i*th neuron of stage *x* with the *j*th neuron of stage y is

$$T_{xi,yj} = -a\delta_{xy}(1-\delta_{ij}) - a - \frac{bd_{xi,yj}(\delta_{(x+1)y} + \delta_{(x-1)y})}{2} \tag{3-5}$$

where

$a\delta_{xy}(1 - \delta_{ij})$ is the inhibitory connection within each stage,

a is the global inhibition,

$bd_{xi,yj}(\delta_{(x+1)y} + \delta_{(x-1)y})/2$ is the strength of the distance metric,

$$\delta_{ij}\ (\text{or } \delta_{xy}) = \begin{cases} 1 & \text{if } i = j\ (x = y) \\ 0 & \text{otherwise,} \end{cases}$$

and the input bias current of *i*th neuron of stage *x* is

$$I_{xi} = an.$$

Therefore, the dynamic programming network can be described by the following differential equation and is conceptually depicted in Figure 3-1:

$$C_{xi}\left(\frac{du_{xi}}{dt}\right) = -\frac{u_{xi}}{R_{xi}} - a\sum_{j\neq i} v_{xj} - a\sum_{y}\sum_{j} v_{yj} + an - \frac{b}{2}\sum_{j}\left(d_{xi,(x+1)j}v_{(x+1)j} + d_{(x-1)j,xi}v_{(x-1)j}\right) \tag{3-6.1}$$

where

$$\frac{1}{R_{xi}} = \frac{1}{\rho_{xi}} + \sum_{yj} T_{xi,yj}. \tag{3-6.2}$$

Moreover, equation (3-2) can be rewritten as

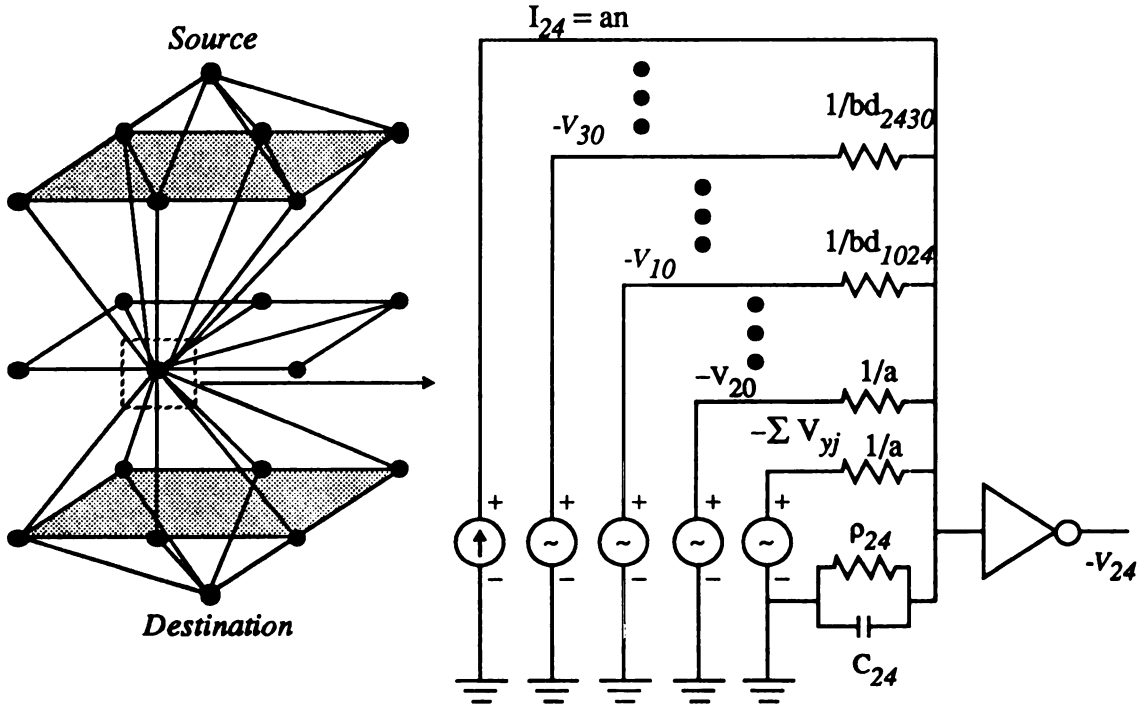


Figure 3-1. A conceptual 3x6 dynamic programming network (full interconnections not shown) and its processing element.

$$E = -\frac{1}{2}\sum_x\sum_i\sum_y\sum_j T_{xi,yj}v_{xi}v_{yj} - \sum_x\sum_i v_{xi}I_{xi} + \sum_x\sum_i \int_{0.5}^{Vxi} \frac{1}{R_{xi}} g_{xi}^{-1}(\xi)\, d(\xi) \cdot \qquad (3\text{-}7)$$

3.2 Network Analysis

To represent equation (3-4) in terms of the variables v_{xi}, it can be written as

$$\frac{dv_{xi}}{dt} = \frac{dv_{xi}}{du_{xi}} \times \frac{du_{xi}}{dt} = \frac{1}{C_{xi}} \times \frac{1}{\dfrac{dg_{xi}^{-1}(v_{xi})}{dv_{xi}}} \times \left(\sum_{yj} T_{xi,yj}v_{yj} - \frac{u_{xi}}{R_{xi}} + I_{xi}\right) = f_{xi}(v) \cdot \qquad (3\text{-}8)$$

Therefore, the network will be considered as an autonomous system since the function f_{xi} is not dependent on time.

3.2.1 Definitions

The following definitions will be used in the remainder of this dissertation.

i) The *domain* and *range* of the sigmoid function g are defined as $U=R^N$ *and* $V=(0, 1)^N$, respectively, where $N = m \times n$. Every vector $v \in V$ can be expressed as $[v_{11},v_{12},\ldots,v_{mn}]^T$, where $0 < v_{xi} < 1$. The zero vector of V is denoted as 0_N. The *closure* and the *boundary* of V are denoted as $\bar{V}$ and ∂V, respectively.

ii) A vector $c \in V$ is said to be a *valid corner state* if c_{xi} is either unity or zero and $\sum_i c_{xi} = 1$ for x=1 to n.

iii) The *valid corner set* C is defined as the set of all valid corner states.

iv) A vector $v \in V$ is said to be in an *h-neighborhood* of a valid corner state c if $\|c - v\|_\infty < h$, where $\|c - v\|_\infty$ is the infinite norm and h is a constant.

v) A vector $v \in V$ is said to be in an *h-neighborhood of C* if $\min_{c \in C} \| c - v \|_\infty < h$.

vi) The set C_h is defined as the set of vectors which are in the h-neighborhood of C, i.e.,

$$C_h = \{ v \in V \mid \min \| c - v \|_\infty < h, \text{ where } c \in V \}.$$

vii) The set Q_c is defined as the set of vectors $q = [q_{11}, q_{12}, \ldots, q_{mn}]^T$ where $q_{xi} = 0$ if $c_{xi} = 0$, and $1 - h < q_{xi} \leq 1$ if $c_{xi} = 1$, for every $c \in C$. The set Q_c is a subset of C_h.

viii) A *vector of convergence*, $v \in V$, is interpreted as a valid solution if it is in the h-neighborhood of C where h is usually chosen to lie between (0, 0.2) [34].

ix) Given $\varepsilon > 0$, the set D_δ is defined as $[\delta, 1 - \delta]^N$ such that $0 \leq \int_{0.5}^{v} g^{-1}(\xi)\, d\xi \leq \varepsilon$ for all $v \in D_\delta$. Note that $0 < \delta < 0.5$ and is a function of the gain λ such that as $\lambda \to \infty$, $\delta \to 0$. Also note that as $\lambda \to \infty$, $D_\delta \to V$.

x) The set $\overline{D}_\delta$ is defined as $V - D_\delta$.

xi) The open set $B(o, \delta)$ is defined as

$$B(o, \delta) = \{ v \in R^N \mid (|v - o| < \delta) \} .$$

xii) A vector $v_0 \in V$ is said to be an ***equilibrium point*** of the system described by equation (3-8) if and only if $f_{xi}(v_0) \equiv 0$ for all x and i.

xiii) The equilibrium point o is said to be *stable* if, for each $\varepsilon > 0$, there exists a $\xi(\varepsilon) > 0$ such that

$$\| v(t_0) - o \| < \xi(\varepsilon) \Rightarrow \| v(t) - o \| < \varepsilon, \quad \forall (t \geq t_0) .$$

xiv) The equilibrium o is *unstable* if it is not stable.

xv) The equilibrium o is asymptotically stable if (1) it is stable, and (2) there exists a number $\xi_1 > 0$ such that

$$\| v(t_0) - o \| < \xi_1 \Rightarrow \| v(t) - o \| \to 0, \quad as \quad (t \to \infty) \cdot$$

xvi) A system on an open set $W \subset R^n$ is said to be a gradient system if

$$\frac{dx}{dt} = -grad\ L(x),$$

where L is a C^2 function, and

$$grad\ L = \left[\frac{\partial L}{\partial x_1} \ , \ldots, \ \frac{\partial L}{\partial x_n} \right]$$

is the gradient vector field.

3.2.2 Analysis of Network Functionality

The following propositions set forth to explain why the formulated network can solve dynamic programming problems. Some similar results have been derived for general Hopfield-type networks [40,41], nevertheless, they are restated for the formulated network in this dissertation for completeness.

Proposition 1. For any initial state $v \in V$, there is a unique solution for the system described by equation (3-8).

Proof: From equation (3-8), $f(v_{xi})$ and $\frac{\partial}{\partial v_{xi}} f(v_{xi})$ are continuous. According to the theory of global uniqueness and existence in systems of differential equations, the network

satisfies a global Lipschitz condition [75]. That is

$$\|f(x) - f(y)\| \leq k\|x - y\|$$

and

$$\|f(x_0)\| \leq h$$

where x, y, and $x_0 \in V$, and k and h are finite constants. Then the network has exactly one solution.

Proposition 2. The vector $o \in V$ is an equilibrium point of the system described by equation (3-8) if and only if

$$\frac{\partial}{\partial v_{xi}} E(v) = 0 \tag{3-9.1}$$

for all x and i.

Proof: Necessary condition:

If o is an equilibrium point of the system described by equation (3-8), then

$$\frac{dv_{xi}}{dt} = f_{xi}(v) = 0 \tag{3-9.2}$$

for all x and i. Therefore, we have

$$\sum_{yj} T_{xi,yj} v_{yj} - \frac{u_{xi}}{R_{xi}} + I_{xi} = \frac{\partial}{\partial v_{xi}} E(v) = 0 \tag{3-9.3}$$

for all x and i.

Sufficient condition:

If

$$\frac{\partial}{\partial v_{xi}} E(v) = 0$$

then

$$\frac{1}{C_{xi}} \times \frac{1}{\dfrac{dg_{xi}^{-1}(v_{xi})}{dv_{xi}}} \times \left(\sum_{yj} T_{xi,yj} v_{yj} - \frac{u_{xi}}{R_{xi}} + I_{xi} \right) = f_{xi}(v) = 0 \cdot$$

for all x and i. Q.E.D.

Proposition 3. The energy function E decreases monotonically along the trajectories and becomes zero only at equilibrium points.

Proof: Since the network is symmetric, that is, $T_{xi,yj}=T_{yj,xi}$, the derivative of E along the trajectories with respect to time can be derived as

$$\frac{dE}{dt} = \sum_x \sum_i \left(\frac{\partial E}{\partial v_{xi}}\right) \left(\frac{dv_{xi}}{du_{xi}}\right) \left(\frac{du_{xi}}{dt}\right) = -\sum_x \sum_i C_i \left(\frac{du_{xi}}{dt}\right)^2 \frac{d}{du_{xi}} g(u_{xi}) \quad . \qquad (3\text{-}10.1)$$

Since g_{xi} is monotonically increasing, dE/dt is less than or equal to zero. dE/dt equals zero only when

$$\frac{du_{xi}}{dt} = -\frac{\partial}{\partial v_{xi}} E(v) = 0 \qquad (3\text{-}10.2)$$

for all x and i. Q.E.D.

Proposition 4. Assume that each equilibrium point of the system described by equation (3-8) is isolated, then there are a finite number of equilibrium points in the system [40,41].

Proof: Since $g_{xi}^{-1}(v_{xi}) \rightarrow +\infty$ as v_{xi} approaches 1, and $g_{xi}^{-1}(v_{xi}) \rightarrow -\infty$ as v_{xi} approaches 0, then

$$\left|\frac{\partial}{\partial v_{xi}} E(v)\right| = \left|-C_{xi}\frac{du_{xi}}{dt}\right| = \left|-\left(\sum_{yj} T_{xi,yj} v_{yj} - \frac{g_{xi}^{-1}(v_{xi})}{R_{xi}} + I_{xi}\right)\right| \tag{3-11}$$

will approach infinity as v approaches the boundary of V. Therefore, there exists a ξ, such that $\frac{\partial}{\partial v_{xi}} E(v) \neq 0$, outside of $C = (0+\xi, 1-\xi)^N$, where $0 < \xi < \frac{1}{2}$. By proposition 2, every equilibrium point must be in the closure of C, $[0+\xi, 1-\xi]^N$. Since the closure of C is compact, and every equilibrium point is isolated, it can be concluded that there are a finite number of equilibrium points in the system. Q.E.D.

Proposition 5. Every isolated equilibrium point of the system described by equation (3-8) is asymptotically stable.

Proof: Since $\frac{dE(v)}{dt} \leq 0$ and $E(v) > 0$ for all v, the energy function E is a strict Liapunov function in some neighborhood of every isolated equilibrium point. Hence, every isolated equilibrium point is asymptotically stable. Q.E.D.

Proposition 6. o is an equilibrium point of the system described by equation (3-8) if and only if o is a local minimum state of the energy function E [40,41].

Proof: Necessary condition:

If o is an equilibrium point, then o is asymptotically stable according to proposition 5. For the purpose of contradiction, assume that o is an equilibrium point and is not a local

minimum state of E. Then there exists a sequence

$$\{o_n\} \subset (0, 1)^N$$

such that

$$E(o_m) < E(o)$$

and

$$0 < |o_m - o| < \frac{1}{m}$$

where $o_m \in \{o_n\}$. Since o is an isolated equilibrium point, there exists an $\varepsilon > 0$ such that there are no equilibrium points in $B(o, \varepsilon) - \{o\}$. For any δ, $\varepsilon > \delta > 0$, choose m such that $\frac{1}{m} < \delta$. Therefore,

$$o_m \in B(o, \delta) - \{o\} \subset B(o, \varepsilon) - \{o\}$$

and o_m is not an equilibrium point. Suppose the solution $\varphi(o_m)$ with initial state o_m converges to an equilibrium point $\hat{o}$. Then

$$E(\hat{o}) < E(o_m) < E(o)$$

by proposition 3. Consequently, $\hat{o}$ is not contained in the region of attraction associated with equilibrium point o. Moreover, o is not asymptotically stable. Hence, the necessary condition is proved by contradiction.

Sufficient condition:

Assume that o is a local minimum state of E and is not an equilibrium point. Since o is a local minimum state of E, there exists an $\varepsilon > 0$ such that

$$E(o) < E(\tilde{o})$$

for all $\tilde{o} \in B(o, \varepsilon)$. Suppose the solution $\varphi(o)$ with initial state o converges to an equilibrium point $\hat{o} \neq o$. Two cases may occur. First, if $\hat{o} \in B(o, \varepsilon)$, then

$$E(\hat{o}) < E(o)$$

Secondly, if $\hat{o} \notin B(o, \varepsilon)$, then there exists a state $o^{\circ} \in B(o, \varepsilon)$ in the trajectory $\varphi(o)$ such that

$$E(o^{\circ}) < E(o) \cdot$$

Contradictions occur in both cases, therefore the sufficient condition is proved. Q.E.D.

In summary, Proposition 1 ensures the existence and uniqueness of the solution. Propositions 2, 3, and 5 guarantee the stability and convergence of the network. The possibility of periodic solutions is avoided by Proposition 4. And Proposition 6 provides a way to find the equilibria from the associated energy function.

3.2.3 Analysis of Minimum States

Many approaches have been developed for analyzing general Hopfield-type networks. For examples, the work by Aiyer, *et al.*, based on the geometry of the subspace set up by the degenerate eigenvalues of the connection matrix, has given an answer to why the Hopfield network frequently converges to invalid solutions when applied to the traveling salesman problem, and suggests some ways to solve the problem reliably [77-79]. Moreover, they have proved that spurious fixed points can occur at any corner of the solution hypercube in the case of content addressable memory, which is consistent with our results obtained by analyzing the network's associated energy function. The work by Abe

has clarified by an eigenvalue analysis, the conditions to converge to a vertex, a point on the edge, or an interior point of the hypercube [80].

Since the network behavior is intimately related to the associated energy function, it is necessary to investigate the characteristics of the energy function in order to explain how the dynamic programming neural network can provide near-optimal solutions. In this dissertation, a two-step analysis is carried out. First, the locations and numbers of the minimum states for different components of the energy function are investigated in the extreme cases. Second, the locations of the minimum states of the energy function using different parameter values are derived.

Let L be a real-valued function on V. For $v \in V$, denote

$$F(L, v) = [f_1(L, v), \ldots, f_n(L, v)]^T_{1 \times N}$$

where

$$f_i(L, v) = \left[\frac{\partial L}{\partial v_{i1}}, \frac{\partial L}{\partial v_{i2}}, \ldots, \frac{\partial L}{\partial v_{im}}\right]_{1 \times m}$$

and

$$G(L, v) = \left[G_{ij}(L, v)\right]_{N \times N}$$

where

$$G_{ij}(L, v) = \begin{bmatrix} \dfrac{\partial f_i(L, v)}{\partial v_{j1}} \\ \dfrac{\partial f_i(L, v)}{\partial v_{j2}} \\ \vdots \\ \dfrac{\partial f_i(L, v)}{\partial v_{jm}} \end{bmatrix}_{m \times m} .$$

The following two applied matrix theorems are also required.

Theorem 1: If A is an $n \times n$ positive definite symmetric matrix, then every principal submatrix of A is also positive definite [81].

Theorem 2: If A is an $n \times n$ positive definite symmetric matrix, then every eigenvalue of A is positive [81].

If a high-gain limit is assumed as is common, then the minimum states of the energy function are very close to those of $\frac{a}{2}E1 + \frac{b}{4}E2$, as the integral term of equation (3-6) can then be neglected[5-8].

The following propositions help to provide a clearer insight into the associated energy function by explicitly showing the minimum states of $E1$ and $E2$.

Proposition 7. There are no local minimum states of $E1$ in the interior of V.

Proof: A state vector v is a local minimum state of $E1$ if the N-vector $F(E1,v)$ is zero and the $N \times N$ matrix $G(E1,v)$, which is the Hessian matrix associated with $E1$, is positive-definite. The first condition leads to N linear equations of the form

$$\frac{\partial E1}{\partial v_{11}} = 2\left(\sum_{j\neq 1} v_{1j} + \sum_{yj} v_{yj} - n\right) = 0, \tag{3-12.1}$$

$$\frac{\partial E1}{\partial v_{12}} = 2\left(\sum_{j\neq 2} v_{1j} + \sum_{yj} v_{yj} - n\right) = 0, \tag{3-12.2}$$

$$\vdots$$

and

$$\frac{\partial E1}{\partial v_{nm}} = 2\left(\sum_{j\neq m} v_{nj} + \sum_{yj} v_{yj} - n\right) = 0 \cdot \tag{3-12.3}$$

The only solution for the N linear equations is

$$\hat{v} = [\hat{v}_{11}, \hat{v}_{12}, \ldots, \hat{v}_{nm}] \in V$$

where

$$\hat{v}_{xi} = \frac{n}{mn + m - 1} \cdot$$

This can be illustrated by the following example with n=2 and m=2. The four linear equations are

$$\frac{\partial E1}{\partial v_{11}} = 2\left(v_{12} + (v_{11} + v_{12} + v_{21} + v_{22}) - 2\right) = 0, \tag{3-13.1}$$

$$\frac{\partial E1}{\partial v_{12}} = 2\left(v_{11} + (v_{11} + v_{12} + v_{21} + v_{22}) - 2\right) = 0, \tag{3-13.2}$$

$$\frac{\partial E1}{\partial v_{21}} = 2\left(v_{22} + (v_{11} + v_{12} + v_{21} + v_{22}) - 2\right) = 0, \tag{3-13.3}$$

and

$$\frac{\partial E1}{\partial v_{22}} = 2\,(v_{21} + (v_{11} + v_{12} + v_{21} + v_{22}) - 2) = 0 \cdot \qquad (3\text{-}13.4)$$

From the first two equations, $v_{11}=v_{12}$ and from the last two equations, $v_{21}=v_{22}$. Substituting v_{12} with v_{11} and v_{22} with v_{21} into the first equation, v_{22} is found to be identical to v_{11}. Therefore, $v_{11}=v_{12}=v_{21}=v_{22}=2/5=n/(mn+m-1)$.

To find out if $\hat{v}$ is a local minimum of $E1$, it is usually necessary to know if $G(E1,v)$ is positive-definite when evaluated at $\hat{v}$. However, from Theorem 1, if any principal submatrix of $G(E1,v)$ is not positive-definite, then $G(E1,v)$ is not positive-definite. The leading principal submatrix of $G(E1,v)$ of order m is

$$G_{11}(E1,v) = \begin{bmatrix} \frac{\partial^2}{\partial v_{11}^2}E1 & \frac{\partial^2}{\partial v_{11}\partial v_{12}}E1 \ldots & \frac{\partial^2}{\partial v_{11}\partial v_{1m}}E1 \\ \frac{\partial^2}{\partial v_{12}\partial v_{11}}E1 & \frac{\partial^2}{\partial v_{12}^2}E1 \ldots & \frac{\partial^2}{\partial v_{12}\partial v_{1m}}E1 \\ \cdots & \cdots & \cdots \\ \frac{\partial^2}{\partial v_{1m}\partial v_{11}}E1 & \frac{\partial^2}{\partial v_{1m}\partial v_{12}}E1 & \frac{\partial^2}{\partial v_{1m}^2}E1 \end{bmatrix}_{m\times m} \qquad (3\text{-}13.5)$$

which can be further calculated to be

$$G_{11}(E1,v) = \begin{bmatrix} 2 & 4 \ldots & 4 \\ 4 & 2 \ldots & 4 \\ \cdots & \cdots & \cdots \\ 4 & 4 \ldots & 2 \end{bmatrix} \cdot \qquad (3\text{-}13.6)$$

Theorem 2 states that if $G_{11}(E1,v)$ is positive-definite, then every eigenvalue of $G_{11}(E1,v)$ must be positive.The eigenvalues of $G_{11}(E1,v)$ satisfy the following condition

$$det\,(G_{11}(E1,v) - \lambda I) = 0, \qquad (3\text{-}13.5)$$

where λ is an eigenvalue of $G_{11}(E1,v)$ and I is the $m \times m$ identity matrix. Therefore, -2 is an eigenvalue of $G_{11}(E1,v)$ because $det(G_{11}(E1,v)+2I)=0$. Also $4(m-1)+2$, which is positive, is another eigenvalue. Since $G_{11}(E1,v)$ is neither positive-definite nor negative-definite, $\hat{v}$ is a saddle point of $E1$. It is thus concluded there are no local minimum states in the interior of V. Q.E.D.

Proposition 8. *v is a valid corner state if and only if v is a global minimum state of $E1$ in V.*

Proof: For every state vector $v \in V$,

$$E1(v) = \sum_x \sum_i \sum_{j \neq i} v_{xi} v_{xj} + \left(\sum_x \sum_i v_{xi} - n\right)^2 \geq 0 . \tag{3-14}$$

If $v \in C$, by the definition of valid corner state, it follows that $\sum_x \sum_i v_{xi} = n$ and $\sum_i \sum_{j \neq i} v_{xi} v_{xj} = 0$. Consequently, $E1(v)=0$ and v is a global minimum state of $E1$. On the other hand, if v is a global minimum state of $E1$, then $E1(v) = 0$. This implies that $\sum_x \sum_i v_{xi} = n$ and $\sum_i \sum_{j \neq i} v_{xi} v_{xj} = 0$. Consequently, v_{xi} is either unity or zero and $\sum_i v_{xi} = 1$ for $x = 1$ to n. Therefore, v is a valid corner state. Q.E.D.

Proposition 9. *The zero state vector 0_N is the unique minimum state of $E2$ in V.*

Proof: For every state vector $v \in V$,

$$E2 = \sum_x \sum_i \sum_j d_{xi,(x+1)j} v_{xi} v_{(x+1)j} + d_{(x-1)j,xi} v_{xi} v_{(x-1)j} \geq 0 . \tag{3-15.1}$$

E2 is zero only at 0_N. Since the first partial derivative of $E2$ with respect to v_{xi},

$$\frac{\partial E2}{\partial v_{xi}} = \sum_j (d_{xi,(x+1)j} v_{(x+1)j} + d_{(x-1)j,xi} v_{(x-1)j}), \qquad (3\text{-}15.2)$$

is greater than zero when $v \neq 0_N$ and equal to zero when $v = 0_N$ for every x and i, it follows that $E2$ is monotonically increasing. Therefore 0_N is the unique minimum state of $E2$. Q.E.D.

Propositions 7 and 8 state that there are exactly n^m valid solutions since $E1$ consists of exactly n^m minimum states. Proposition 9 shows that the zero state vector 0_N is the unique minimum state of $E2$. These three propositions together imply that the minimum states of E will be close to corner states if $\frac{a}{2}E1$ dominates over $\frac{b}{4}E2$; they will be close to 0_N if the reverse is true.

Proposition 10. There are no local minimum states of $\hat{E} = \frac{a}{2}E1 + \frac{b}{4}E2$ in the interior of V.

Proof: The proof is similar to that of Proposition 7. We have that

$$f_1(\hat{E}, v) = \left[\frac{\partial \hat{E}}{\partial v_{11}}, \frac{\partial \hat{E}}{\partial v_{12}}, \ldots, \frac{\partial \hat{E}}{\partial v_{1m}}\right] \qquad (3\text{-}16.1)$$

where

$$\frac{\partial \hat{E}}{\partial v_{1i}} = a\left(\sum_{j \neq i} v_{1j} + \sum_{yj} v_{yj} - n\right) + \frac{b}{4}\sum_j (d_{1i,2j} v_{2j} + d_{00,1i} v_{00}), \qquad (3\text{-}16.2)$$

and v_{00} is the source node and $d_{00,1i}$ is the distance between the source node and node j of the first stage. It can be shown that $G_{11}(\hat{E}, v)$, the leading principal submatrix of $G(\hat{E}, v)$ of order m, is equal to $a \times G_{11}(E1, v)$. Since $G_{11}(E1,v)$ is not positive-definite, $G_{11}(\hat{E}, v)$

is not positive-definite. Therefore, Proposition 10 is concluded. Q.E.D.

According to Proposition 10, the local minimum states of E, i.e., the equilibrium points, are in the set $\bar{B}_\delta$. This means the equilibrium points are very close to the boundary of V when a high-gain limit is assumed. Finally, due to the gradient system property, which states that every equilibrium point is stable [76], the following corollary must be true.

Corollary 1. The network does not possess periodic trajectories.

3.2.4 Location of Minimum States

In this section, the possible locations of the minimum states of the associated energy function of the dynamic programming neural network will be discussed. With every $d_{xi,(x+1)j}$ fixed, the minimum states of the network can be approximately placed in a neighborhood of C such that the minimum states can be interpreted as a valid path with a locally minimum cost by appropriately adjusting parameters a and b. Assume that b is fixed, three cases may occur relating to different values of a.

Case 1: $a \to \infty$.

The energy function E is approximately equal to $\frac{a}{2}E1$ and the minimum states will be determined by $E1$ exclusively. Therefore, the minimum states are in the neighborhood of C and the number of minimum states is equal to n^m. However, the corresponding paths, which can be considered as randomly selected paths, may be unacceptably far from optimal.

Case 2: a=0.

Since the minimum state is determined by $E2$, there is a unique minimum state, i.e., the zero state.

Case 3: $0 < a < \infty$.

In this case, a qualitative discussion is appropriate. As a decreases, the minimum states may move away from corner states in C. The following proposition indicates the

possible locations of the minimum states if they are in the neighborhood of C.

Proposition 11. Assume there is at most one minimum state vector w in the h-neighborhood of a valid corner state c where

$$h \leq \frac{2a + bd_{min}}{2a(n+1) + bd_{min}} \tag{3-17}$$

and d_{min} is the minimum value of metric data. Then the minimum state vector w belongs to Q_c which has been defined in Section 3.1. In other words, w_{xi}=0 if c_{xi}=0, and $1 - h \leq w_{xi} \leq 1$ if c_{xi}=1 for all x and i.

Proof: Every vector v in the h-neighborhood of c can be decomposed as

$$v = v_c + v_{\bar{c}}$$

where

$$v_c = [c_{11}v_{11}, c_{12}v_{12}, \ldots, c_{nm}v_{nm}] \in Q_c$$

and

$$v_{\bar{c}} = [(1 - c_{11})v_{11}, \ldots, (1 - c_{nm})v_{vm}] \cdot$$

Then, $E1(v)$ and $E2(v)$ can be expressed as follows.

$$E1(v) = \sum_x \sum_i \sum_{j \neq i} [c_{xi}v_{xi} + (1 - c_{xi})v_{xi}] \times [c_{xj}v_{xj} + (1 - c_{xj})v_{xj}] + \left\{ \left[\sum_x \sum_i c_{xi}v_{xi} + (1 - c_{xi})v_{xi} \right] - n \right\}^2 \tag{3-18.1}$$

and

$$E2(v) = \sum_x \sum_i \sum_j d_{xi,(x+1)j} [c_{xi} v_{xi} + (1 - c_{xi}) v_{xi}] \times$$
$$[c_{(x+1)j} v_{(x+1)j} + (1 - c_{(x+1)j}) v_{(x+1)j}] +$$
$$d_{(x-1)j,xi} [c_{xi} v_{xi} + (1 - c_{xi}) v_{xi}] [c_{(x-1)j} v_{(x-1)j} + (1 - c_{(x-1)j}) v_{(x-1)j}] .$$

(3-18.2)

Therefore, $E(\mathbf{v})$ can be derived as

$$E(v) = \frac{a}{2} E1(v_c) + \frac{b}{4} E2(v_c) + S_1(v) + S_2(v) \tag{3-18.3}$$

where

$$E1(v_c) = \sum_x \sum_i \sum_{j \neq i} c_{xi} v_{xi} c_{xj} v_{xj} + \left(\sum_x \sum_i c_{xi} v_{xi} - n\right)^2, \tag{3-18.4}$$

$$E2(v_c) = \sum_x \sum_i \sum_j d_{xi,(x+1)j} c_{xi} v_{xi} c_{(x+1)j} v_{(x+1)j} + d_{(x-1)j,xi} c_{xi} v_{xi} c_{(x-1)j} v_{(x-1)j}, \tag{3-18.5}$$

$$S_1(v) = \frac{a}{2} \sum_x \sum_i \sum_{j \neq i} (1 - c_{xi}) v_{xi} (1 - c_{xj}) v_{xj} + \frac{a}{2} \left(\sum_x \sum_i (1 - c_{xi}) v_{xi}\right)^2 +$$
$$\frac{b}{4} \sum_x \sum_i \sum_j d_{xi,(x+1)j} (c_{xi} v_{xi} (1 - c_{(x+1)j}) v_{(x+1)j} + (1 - c_{xi}) v_{xi} (1 - c_{(x+1)j}) v_{(x+1)j})$$
$$+ d_{(x-1)j,xi} (c_{xi} v_{xi} (1 - c_{(x-1)j}) v_{(x-1)j} + (1 - c_{xi}) v_{xi} (1 - c_{(x-1)j}) v_{(x-1)j}),$$

(3-18.6)

and

$$S_2(v) = a\left(\sum_x\sum_i c_{xi}v_{xi} - n\right)\left(\sum_x\sum_i (1-c_{xi})\,v_{xi}\right) + \frac{a}{2}\sum_x\sum_i\sum_{j\neq i} c_{xi}v_{xi}(1-c_{xj})\,v_{xj}$$
$$+ \frac{a}{2}\sum_x\sum_i\sum_{j\neq i} c_{xj}v_{xj}(1-c_{xi})\,v_{xi} + \frac{b}{4}\sum_x\sum_i\sum_j d_{xi,\,(x+1)j}c_{(x+1)j}v_{(x+1)j}(1-c_{xi})\,v_{xi}$$
$$+ \frac{b}{4}\sum_x\sum_i\sum_j d_{(x-1)j,\,xi}c_{(x-1)j}v_{(x-1)j}(1-c_{xi})\,v_i \cdot \tag{3-18.7}$$

Each term of $S_1(v) \geq 0$ is greater than or equal to zero. Since v is in the h-neighborhood of c, there exists one and only one k for every $x = 1$ to n, where $1 \leq k \leq m$, $c_{xk} = 1$, and $c_{xj} = 0$ if $j \neq k$, such that $1 - h \leq v_{xk} \leq 1$. Then we have

$$\sum_i c_{xi}v_{xi} = v_{xk} \geq 1 - h, \tag{3-18.8}$$

and

$$\sum_x\sum_i\sum_{j\neq i} c_{xi}v_{xi}(1-c_{xj})\,v_{xj} = \sum_x\sum_i\sum_{j\neq i} c_{xj}v_{xj}(1-c_{xi})\,v_{xi}$$
$$= \sum_x\sum_i\sum_j c_{xj}v_{xj}(1-c_{xi})\,v_{xi} \geq (1-h)\left(\sum_x\sum_i (1-c_{xi})\,v_{xi}\right) \cdot \tag{3-18.9}$$

Therefore, equation (3-18.7) can be rewritten as

$$S_2(v) \geq \sum_x\sum_i (1-c_{xi})\,v_{xi}\left[a(1-h) + \frac{bd_{min}}{2}(1-h) - anh\right] \cdot \tag{3-18.10}$$

Furthermore, if

$$h \leq \frac{2a + bd_{min}}{2a(n+1) + bd_{min}},$$

then $S_2(v)$ is greater than or equal to zero. Therefore, $E(v) > E(v_c)$ for every $v = v_c + v_{\bar{c}}$ and $v_{\bar{c}} \neq 0_N$. Thus, if w is the only minimum state of E in C_h, then w must be in Q_c. Q.E.D.

With certain conditions, it has been shown that the assumption used in Proposition 5 is valid [45]. According to Proposition 5, the associated energy function can be expressed as

$$E(v) = E(v_c) = \frac{a}{2}E1(v_c) + \frac{b}{4}E2(v_c) \cdot \tag{3-19}$$

Therefore, if the minimum states are in the h-neighborhood of C, they can be obtained from equation (3-19) with only n unknown variables, instead of the original

$$E(v) = \frac{a}{2}E1(v) + \frac{b}{4}E2(v)$$

which consists of $n \times m$ variables. This reduction will be very helpful in finding the minimum states of the associated energy function.

For a minimum state v, suppose that $v_x = max(v_{xi})$ for i=1 to n, O is the output state vector, $\hat{P}_x$ is the processing element associated with v_x in stage x, $\hat{d}_{x,x+1}$ is the metric data between $\hat{P}_x$ and$\hat{P}_{x+1}$, and $\hat{d}_{x-1,x}$is the metric data between$\hat{P}_x$ and $\hat{P}_{x-1}$. Without loss of generality, assume that g_{xi}=g, R_{xi}=R, and C_{xi}=C for every x and i. Also redefine α=a/C and β=b/C.

Better quality paths result when a is small [39]. Therefore, it is desirable to use a smaller value of a and also keep the minimum states in the neighborhood of C so that the minimum states can be regarded as valid and quality solutions. However as a is decreased to a modest value, two cases may occur. First, the ideal case where the output state vector O stays in the h-neighborhood of C could occur when the sum of $\hat{d}_{x,x+1}$ and $\hat{d}_{x-1,x}$ are very close to each other for all x. Second, O may fall outside the h-neighborhood of C if there is at least one $\hat{P}_x$ where the sum of $\hat{d}_{x,x+1}$ and $\hat{d}_{x-1,x}$ is very large compared to others. To prevent the second case from occurring, a maximum value $\tilde{v}$ which satisfies the following conditions is set to limit v_x.

For the particular processing element $\hat{P}_x$ in stage x,

others. To prevent the second case from occurring, a maximum value $\tilde{v}$ which satisfies the following conditions is set to limit v_x.

For the particular processing element $\hat{P}_x$ in stage x,

$$\frac{du}{dt} = -\hat{u} - \alpha n\tilde{v} - \beta\tilde{v}d_{avg} + \alpha n > 0, \tag{3-20.1}$$

where $\hat{u} = g^{-1}(\tilde{v})$, then

$$\tilde{v} < \frac{\alpha n - \hat{u}}{\alpha n + \beta d_{avg}} . \tag{3-20.2}$$

For the other processing elements in stage x,

$$\frac{du}{dt} = u_{max} - \alpha\tilde{v} - \alpha n\tilde{v} + \alpha n - \beta\tilde{v}d_{min} < 0 . \tag{3-21.1}$$

Therefore,

$$\tilde{v} > \frac{\alpha n + u_{max}}{\alpha(n+1) + \alpha d_{min}} \tag{3-21.2}$$

where u_{max} is the maximum allowable value of u due to circuit considerations, d_{avg} is the average of the local distances $d_{xi,(x+1)j}$s, and d_{min} is the minimum of the local distances.

These two conditions are derived from the network dynamical equation (3-5.1) with an assumption that the sum of $\hat{d}_{x,x+1}$ and $\hat{d}_{x-1,x}$ is less than $2d_{avg}$. The first condition guarantees that every $\hat{P}_x$with output value less than $\tilde{v}$ will increase its output value until reaching $\tilde{v}$. The second condition guarantees that the output values of the inactive processing elements will decrease until they become very close to zero. Ultimately, for each stage x, the trajectory will be forced to reach the state where only $\hat{P}_x$ has an output value equal to $\tilde{v}$ and all others are very close to zero.

From equation (3-20.2), with $\vartheta = 1 - h$, a lower limit for α can be derived as

$$\alpha > \frac{\beta d_{avg}(1-h) + g^{-1}(1-h)}{nh} . \tag{3-22}$$

As a is decreased below the value given in equation (3-22), the minimum states

move too far away from the h-neighborhood of C such that they can no longer be regarded as valid paths.

3.2.5 Example

The following example demonstrates the theoretical results presented in this section. Figure 3-2a shows a simple problem which has an optimal path with a total distance of 2 units. The energy function in the form of equation (3-6) is composed of

$$E1 = v_{11}^2 + v_{12}^2 + 4v_{11}v_{12} - 2v_{11} - 2v_{12} + 1, \tag{3-23.1}$$

and

$$E2 = 2v_{11} + 18v_{12}. \tag{3-23.2}$$

There are two minimum states (1, 0) and (0, 1) in $E1$ according to Figure 3-2b. Additionally, the saddle point of $E1$ is (1/3, 1/3). The state (0, 0) is the unique minimum state of $E2$ shown in Figure 3-2c. Choosing a=50 and b=4, Figure 3-2d shows the associated energy function possesses only one minimum state which is close to the corner state (1, 0). The accurately calculated minimum state is (0.96, 0) as determined by equation (3-19).

3.3 Summary

This chapter presented a dynamic programming artificial neural network that can rapidly and robustly provide a near-optimal solution for the DPP. Since dynamic programming is a very useful technique for solving optimal control problems, the proposed network is quite practical.

The core of this chapter is an analytic exploration of the proposed dynamic programming neural network. First of all, propositions related to the functionality of the network were proved. Secondly, a two-step energy function analysis was carried out in

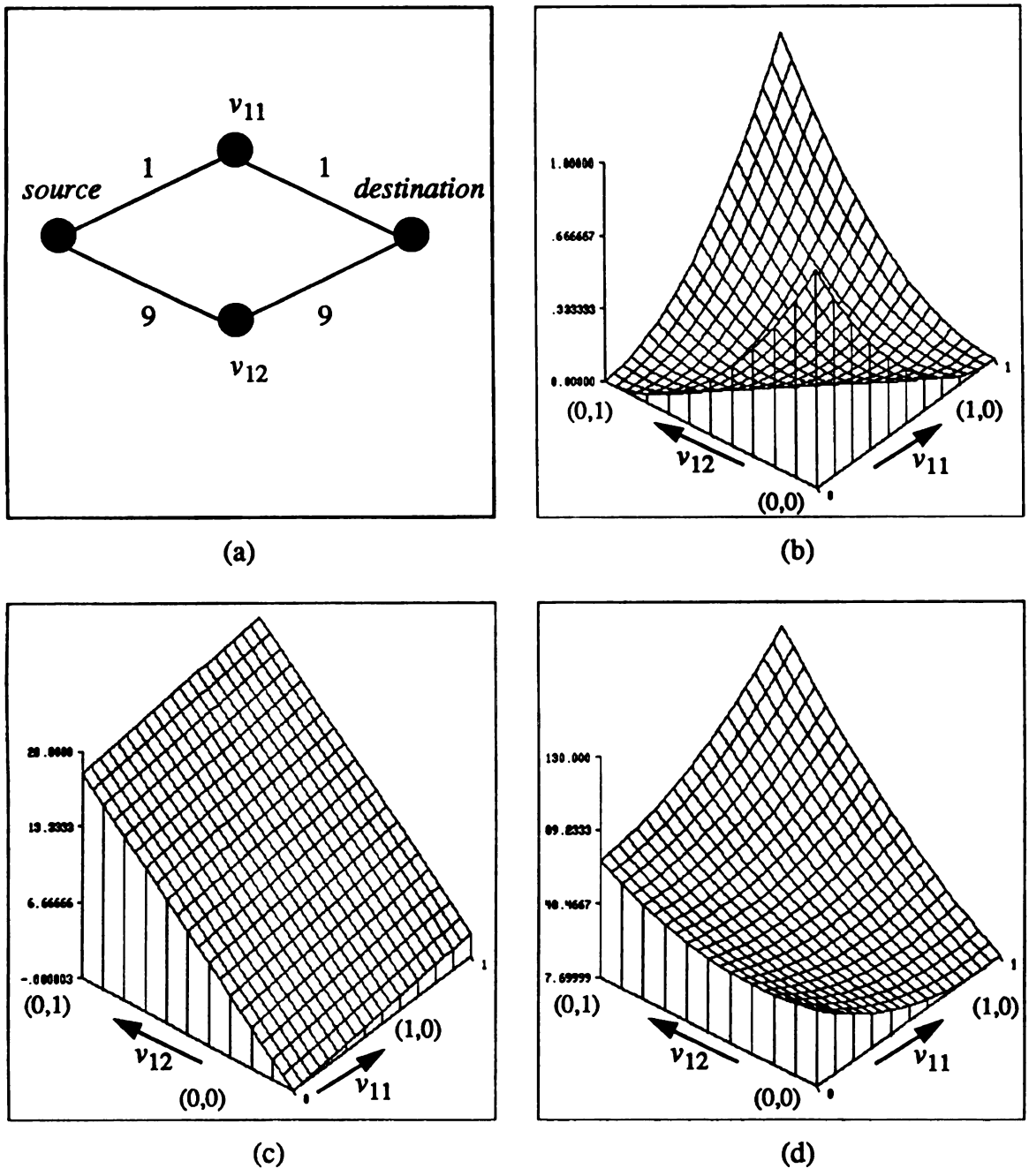


Figure 3-2. a) Data for demonstrating the example, b) Energy surface for $E1$, c) Energy surface for $E2$, and d) Energy surface for $E=25E1+E2$.

order to explain how the dynamic programming neural network can provide near-optimal solutions. Each component of the associated energy function has been discussed in the extreme cases. With $a \to \infty$, the minimum states are the valid corner states representing valid solutions. With $a = 0$, the zero state is the unique minimum state. Then, the possible locations of the minimum states of the associated energy function for different parameter values a and b have been derived. We also have shown that a must be sufficiently large to ensure that the minimum states reside in the regions which can be regarded as valid solutions. Through the minimum states, the network behavior can be predicted and the near-optimal solutions can be obtained. The analysis developed in this paper can be further extended to other networks, such as the TSP Hopfield-Tank network.

It is more interesting to investigate the region of attraction of every stable equilibrium point so that the initial conditions can be appropriately given to obtain the desired solutions. How the parameters a and b affect the number of minimum states of the associated energy function is another issue to be studied further.

Chapter 4

4.0 Network Simulation

This chapter describes a simulation of the formulated network to validate the analysis in Chapter 3. Three items are studied by the simulation: the relationship between the quality of solutions and the parameters of the associated energy function, the network behavior due to random errors in connection weights, and the investigation of the basins of attraction of the network's equilibria. Results show that the formulated network can provide a near-optimal solution during an elapsed time of only a few characteristic time constants of the circuit for different problems with sizes as large as 64 stages with 64 states in each stage. An application of the proposed algorithm to an optimal control problem is also presented.

4.1 Network Simulation

Since the network is described by a set of first order differential equations, it can be simulated by solving the associated differential equations with a numerical analysis techniques running on a conventional computers. Simulation programs written in C language for the formulated network have been developed.

4.1.1 Simulation Algorithm

For convenience, rewrite the network equation (3-6.1) as

$$C_{xi}\left(\frac{du_{xi}}{dt}\right) = -\frac{u_{xi}}{R_{xi}} - a\sum_{j\neq i} v_{xj} - a\sum_{y}\sum_{j} v_{yj} + an$$
$$-\frac{b}{2}\sum_{j}\left(d_{xi,(x+1)j}v_{(x+1)j} + d_{(x-1)j,xi}v_{(x-1)j}\right) \qquad (4\text{-}1.1)$$

where

$$\frac{1}{R_{xi}} = \frac{1}{\rho_{xi}} + \sum_{yj} T_{xi,yj}. \qquad (4\text{-}1.2)$$

To simplify the simulation, we assume that $g_{xi} = g$, $R_{xi} = R$, and $C_{xi} = C$ for all x and i. The speed of convergence may be altered by the use of different values of electronic components in the neurons. However, the solutions will remain the same since the network is a gradient system and the locations of equilibria only depends on the connection weights. The gain of the sigmoid working function $1/u_o$ is set at 50 as a high gain assumption. The inputs of the analog processing elements u_{xi} are limited in the range of [-5, 5] since g(5)= 1/2(1+tanh(250)) is very close to 1 and g(-5) is very close to 0. Finally, without loss of generality, the value of RC is set to 1. Define α=a/C and β=b/C. Then, equation (4-1.1) can be rewritten as

$$(du_{xi}/dt) = -u_{xi} - \alpha\sum_{j\neq i} v_{xj} - \alpha\sum_{yj} v_{yj} -$$
$$\beta/2\sum_{j}\left(d_{xi\,(x+1)j}v_{(x+1)j} + d_{(x-1)j\,xi}\,v_{(x-1)j}\right) + \alpha n \qquad (4\text{-}2)$$

Although the network is considered to be analog in nature, the state of the processing elements will be defined as active, inactive, or undefined depending on their output values. As a conservative circuit convention, the processing elements with output

values greater than 0.90 are considered as active, those with output values less than 0.10 are considered as inactive, while those with output values in-between are undefined. Thus a valid path is defined as one which corresponds to a state with one and only one value of output greater than 0.90 and the others less than 0.10 in each stage.

Four major steps are followed:

Step 1. Randomly generate local distances of $d_{xi\,(x+1)j}$.

Step 2. Set appropriate values for α and β (see below).

Step 3. Set the initial values u_{xi} equal to the sum of u_{oo} and δu_{xi} for all x and i, where the disturbance δu_{xi} is randomly chosen and uniformly distributed in $[-0.1u_o, 0.1u_o]$. The voltage u_{oo} is a constant such that $m*g(u_{oo})$ is equal to unity, where m is the number of states within one stage.

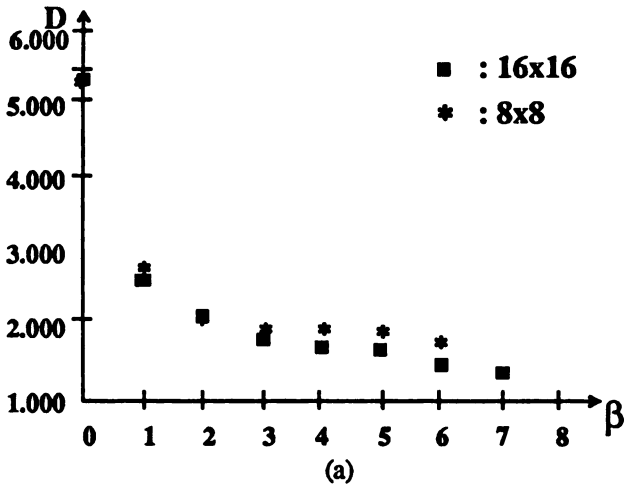
Step 4. Solve equation (7) for all x and i, where

$$V_{xi} = g(u_{xi}) = (1/2)(1 + \tanh(u_{xi} / u_o)).$$

4.1.2 Quality of Solutions and Function Parameters

An experiment has been done to find an appropriate value for β for 8x8, 16x16 and 32x32 dynamic programming problems. A specified set of 10 examples in which the local distances are selected from the integer set of {1, 3, 5, 7, 9} has been simulated with variable β. The average normalized path length D, which is equal to the sum of the selected paths divided by the number of simulations, i.e., 10, and the number of stages plus one, is used in Figure 4-1.a and 4-1.b. The relation between D and β changing from 0 to 7 for the 8x8 and 16x16 problems with α set to 25 is shown in Figure 4-1.a. The network may provide invalid results when β is greater than 6 in this case. Figure 4-1.b depicts D vs. β changing from 0 to 3 for the 32x32 problems with α equal to 5. Invalid results may occur if β is greater than 2.5. It is important to notice that when structure is enforced by using a small value of β, selected paths are of poor quality and when cost minimization is attempted by using a large value of β, invalid paths result. Figures 4-1.a and 4-1.b give a broad choice for parameter β.

Even though the local distances have been selected from the integer set of {1, 3, 5,



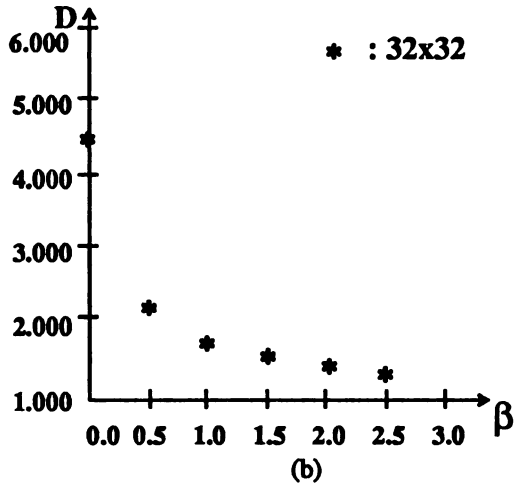


Figure 4-1. a) The relation between the average normalized path length D and different values of β for 8x8 and 16x16 DPPs and b) The relation between the average normalized path length D and different values of β for 32x32 DPPs.

7, 9} i

the co

{1, 3,

0.2,...

Table

found

7, 9} in the previous experiment, it is not absolutely necessary to do so. Table 4-1 shows the comparison of results between simulations with local distances selected from the set of {1, 3, 5, 7, 9} (group I) and simulations with local distances selected from the set of {0.1, 0.2,....9.8, 9.9} (group II). The percentiles of the same size problems are almost the same. Table 4-1 also reveals a promising property. That is, shorter average normalized lengths are found more easily for larger problem sizes.

size / group	8x8	16x16	32x32
group I	2.00	1.85	1.61
group II	1.74	1.44	1.10

Table 4-1. Comparison of D between two groups of simulations.

D	1.00	1.50	2.00	2.50	3.00
P	0.00	1E-5	3E-4	2E-3	2E-2

Table 4-2. Lookup table of P for some values of D.

The proposed algorithm can provide better outputs if the local distances are randomly selected from a set whose components are uniformly distributed in a positive real region. For those cases where the local distances are not uniformly distributed, a method of normalization can be utilized so that good results can be obtained. In the following simulations, local distances was selected from the set {1, 3, 5, 7, 9} for simplicity.

To ascertain the true quality of a selected path, its percentile among all possible paths should be known. However, due to the huge computation time needed to obtain the percentile of a selected path even for medium size problems, the average normalized path length can not be calculated in each simulation. Given D, the percentile P of a selected path

can be approximated by the values given in Table 4-2 which has been constructed from 10^6 randomly generated paths.

4.1.3 Neural Network Solutions for Different Problems

A 6x6 DPP is presented to demonstrate the behavior of the dynamic programming network. This example shows how the dynamic programming network indeed converges to an optimal solution. Table 4-3 gives the local distances for the 6x6 DPP whose optimal path total length is known to be 13. Table 4-4a gives the neuron initial outputs and Tables 4-4b, 4-4c and 4-4dd show the neuron outputs after 25, 50, and 100 iterations, respectively. It can be seen that the network reaches a nearly stable state after 50 iterations. Figures 4-2a, b, c, and d give a graphic view of the time evolution of the dynamic programming network. It is noticed that there are two optimal solutions. The first one, which is the one found by the neural network algorithm is 3-1-1-3-3-1-1, and the other is 1-1-3-3-1-1-3.

To show that the proposed algorithm can provide near-optimal solutions robustly, many different DPP instances have been simulated. Table 4-5 lists the simulation results. For those instances with sizes less than 32x32, each has been simulated 20 times with random initial conditions and local distances. Other cases have been simulated 10 times. The average percentiles P are obtained from Table 4-2. The asterisked percentiles are the exact values computed by exhaustive comparisons. The results in Table 4-5 show that indeed the formulated network produces near-optimal solutions for these problems. For example, P is less than 3×10^{-3} and D is about 1.85 which is much less than the average value 5 for the 16x16 problems.

4.1.4 Basins of Attractions

Given a 4x8 network, shown in Figure 4-3, which consists of two global minimum cost paths with total distance = 0, denote the first minimum path as path A which passes through state 0 of each stage and is tagged as code A = (0, 0, 0, 0, 0, 0, 0, 0), and the second minimum path as path B which passes through state 3 of each stage and is tagged as code

$d_{00\,1j}$	
$j=0$	7
1	9
2	[3]
3	5
4	3
5	1

$d_{1i\,2j}$	j=0	1	2	3	4	5
$i=0$	1	7	1	9	7	9
1	7	9	5	5	7	9
2	5	1	[1]	3	3	1
3	5	3	3	3	9	0
4	3	9	7	9	9	3
5	1	9	3	5	9	5

$d_{2i\,3j}$	j=0	1	2	3	4	5
$i=0$	9	9	3	7	5	9
1	5	3	3	7	1	7
2	1	5	9	3	[1]	7
3	7	5	1	7	3	9
4	3	7	9	5	5	1
5	3	3	9	7	1	7

$d_{3i\,4j}$	j=0	1	2	3	4	5
$i=0$	9	5	5	3	5	5
1	9	5	9	1	3	1
2	9	3	9	9	9	5
3	1	5	1	3	3	5
4	3	7	3	7	[3]	3
5	7	5	1	1	7	1

$d_{4i\,5j}$	j=0	1	2	3	4	5
$i=0$	5	5	5	9	7	1
1	3	9	3	5	1	5
2	9	1	3	9	5	7
3	7	1	3	9	1	9
4	5	9	[3]	7	3	1
5	7	9	9	5	1	9

$d_{5i\,6j}$	j=0	1	2	3	4	5
$i=0$	1	7	1	7	1	5
1	3	9	7	9	1	5
2	5	1	9	[1]	1	1
3	9	9	3	7	9	5
4	1	9	7	9	5	7
5	3	9	7	3	5	1

$d_{6i\,70}$	
$i=0$	3
1	7
2	3
3	[1]
4	9
5	7

Table 4-3. Local distances and path found by the algorithm for the 6x6 DPP. The total length of the selected path = 3 + 1 +1 + 3 + 3 + 1 + 1 = 13.

V_{xi}	*i=0*	*1*	*2*	*3*	*4*	*5*
x=1	0.191	0.147	0.165	0.144	0.193	0.143
2	0.149	0.195	0.195	0.190	0.187	0.181
3	0.174	0.144	0.149	0.182	0.161	0.144
4	0.179	0.148	0.188	0.153	0.195	0.168
5	0.168	0.187	0.195	0.168	0.146	0.171
6	0.162	0.178	0.178	0.170	0.163	0.175

(a)

V_{xi}	*i=0*	*1*	*2*	*3*	*4*	*5*
x=1	0.178	0.029	0.411	0.046	0.087	0.046
2	0.026	0.081	0.492	0.043	0.039	0.074
3	0.098	0.038	0.013	0.257	0.305	0.031
4	0.066	0.048	0.162	0.076	0.270	0.052
5	0.056	0.040	0.482	0.015	0.039	0.101
6	0.060	0.003	0.135	0.788	0.000	0.003

(b)

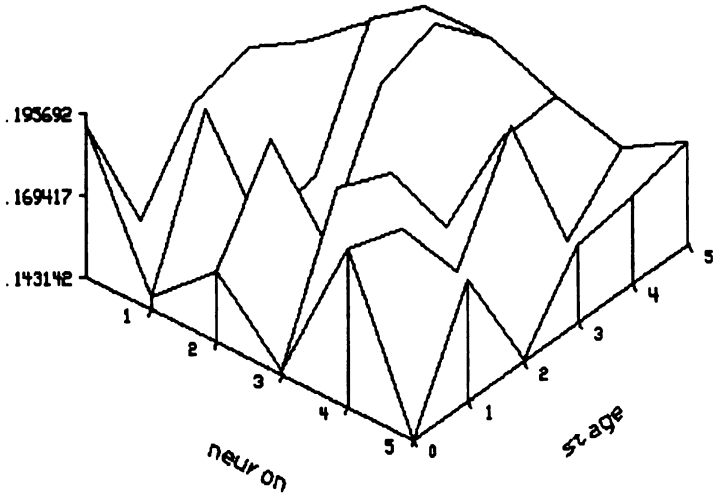
V_{xi}	*i=0*	*1*	*2*	*3*	*4*	*5*
x=1	0.002	0.000	<u>0.985</u>	0.000	0.000	0.000
2	0.000	0.000	<u>0.993</u>	0.000	0.000	0.000
3	0.001	0.000	0.000	0.033	<u>0.821</u>	0.000
4	0.000	0.000	0.011	0.000	<u>0.844</u>	0.000
5	0.000	0.000	<u>0.991</u>	0.000	0.000	0.000
6	0.000	0.000	0.000	<u>0.999</u>	0.000	0.000

(c)

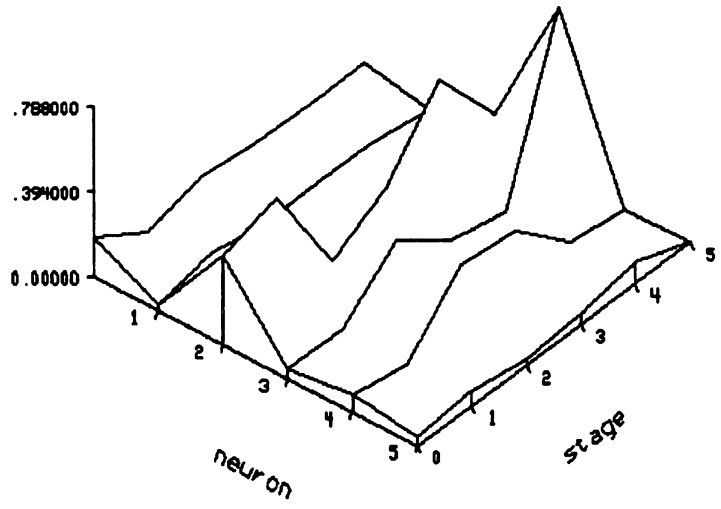
V_{xi}	*i=0*	*1*	*2*	*3*	*4*	*5*
x=1	0.000	0.000	<u>0.999</u>	0.000	0.000	0.000
2	0.000	0.000	<u>0.999</u>	0.000	0.000	0.000
3	0.000	0.000	0.000	0.000	<u>0.967</u>	0.000
4	0.000	0.000	0.000	0.000	<u>0.927</u>	0.000
5	0.000	0.000	<u>0.998</u>	0.000	0.000	0.000
6	0.000	0.000	0.000	<u>0.999</u>	0.000	0.000

(d)

Table 4-4. a) Initial neuron outputs for the 6x6 DPP; b-d) Neuron outputs after 25, 50, and 100 iterations, respectively.

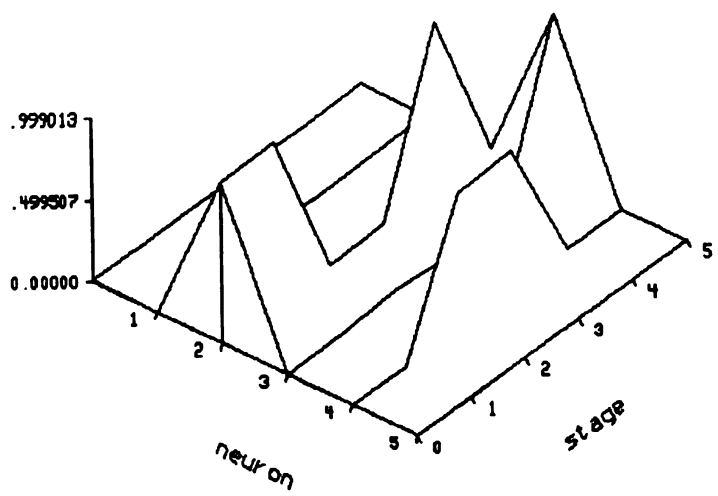


(a)



(b)

Figure 4-2. a) Initial neuron outputs for the 6x6 DPP; b-d) Neuron outputs after 25, 50, and 100 iterations, respectively.



(c)

(d)

Figure 4-2. (cont'd)

# of stages	# of states	D	P
2	2	3.25	0.05*
4	4	3.12	0.03*
8	8	2.00	~ 3E-4
16	16	1.85	3E-4>P>1E-5
32	32	1.61	3E-4>P>1E-5
64	64	1.39	1E-5>P>0.0
16	2	3.21	0.02*
16	4	2.98	0.005*
16	8	1.85	3E-4>P>1E-5
16	32	1.79	3E-4>P>1E-5
2	16	1.53	0.02*
4	16	1.60	0.004*
8	16	1.76	3E-4>P>1E-5

Table 4-5. Simulation results for DPPs with different sizes. The parameter pair (α, β) has been chosen as (50, 3) for instances with the number of stages less than 8, (5, 1) for instances with the number of stages greater than 16, and (25, 3) for instances with the number of stages in between.

B = (1, 1, 1, 1, 1, 1, 1, 1). 2^8 = 256 different codes are used to define initial conditions as follows. If the *i*th bit of the initial condition code is 0, then the neuron 0 of the ith stage is given 0.9 as an initial output value and the other neurons of the ith stage are given 0.1 as their initial value. On the other hand, if the *j*th bit of the initial condition code is 1, then the neuron 3 of the *j*th stage is given 0.9 as a initial output value and the other neurons of the *j*th stage are given 0.1 as their initial value.

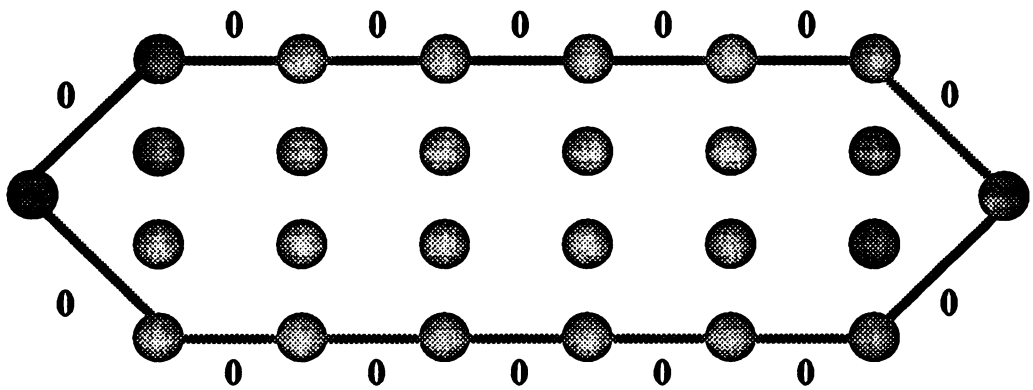


Figure 4-3. A problem consisting two minimum paths shown in thick lines. The local distances for the two minimum paths are all zero, other local distances, not shown, are all set to 50.

Table 4-6 shows the number of attracted initial conditions to global minimum states and local minimum states of the network with $\alpha = 50$ and $\beta = 4$, where H is the Hamming distance between code A or B and the initial condition codes. As can be seen in the table, the number of attracted initial conditions increases when *h* which is defined in Chapter 3 is increased. The case of $h = 0.20$ gives the largest number (184) of attracted initial conditions.

To understand how the parameters α and β affect the basins of attraction of the global minimum states, the following experiment has been done using the previous example. With *h* and α fixed to 0.10 and 50, respectively, β is changed from 1 to 5. For every value of β, 2^8 = 256 initial conditions have been fed to the network. The numbers of

attracted initial conditions of different Hamming distances to code A and B for each value of β is listed in Table 4-7. It can be seen that the number of attracted initial conditions, which may be used as an index of the basins of attraction, increases when β is increased. However, the network may converge to local minimum states as β becomes greater than 4.

Number of attracted init. cond's.	Global minima		Local minima
	$0 \le H \le 4$	$5 \le H \le 6$	
h = 0.01	91	0	165
h = 0.05	107	0	149
h = 0.10	107	0	149
h = 0.15	131	0	125
h = 0.20	184	41	31

Table 4-6. Number of attracted initial conditions to global and local minima with different initial conditions.

β \ H / code		0	1	2	3	4	5
1	A	1	0	0	0	0	0
	B	1	0	0	0	0	0
2	A	1	0	0	0	0	0
	B	1	0	0	0	0	0
3	A	1	8	0	0	0	0
	B	1	8	0	0	0	0
4	A	1	8	22	18	3	0
	B	1	8	22	20	4	0
5	A	1	8	28	39	24	2
	B	1	8	28	49	41	7

Table 4-7. Number of attracted initial conditions to global minima with different values of β.

4.1.5 Network Behavior due to Connection Weight Errors

The Hopfield-Tank networks have been shown to be gradient systems only when the networks are symmetric [14,15]. The connection weights may not be as accurate as desired in a real hardware implementation, therefore the networks are not exactly symmetric. However, it is believed that sufficiently small changes of the connection weights will not alter the qualitative properties of the networks [83].

In this simulation, the previous simple example is used again. With $\alpha = 50$, $\beta = 4$, and initial condition code being set either (0, 0, 0, 0, 0, 0) or (1, 1, 1, 1, 1, 1), simulation shows that these two minimum states are still preserved if the errors of connection weights are less than 5% of the original values. As the errors increases over 5%, the network may become unstable or the minimum states may move away from their original locations such that they can not be considered as minimum cost paths any longer. It is worthy noting that the allowable error in connection weights for the minimum states to remain in the valid regions heavily depends on the problems themselves. In general, the smaller the corresponding total path of a minimum state is, the more likely the minimum state can be preserved in its associated valid region.

4.2 Application to an Optimal Control Problem

Consider a continuous system described by the first order differential equation

$$\frac{dx(t)}{dt} = -20x(t) + u(t) \tag{4-3}$$

where $x(t)$ and $u(t)$ are the state and control variables, respectively. The performance measure to be minimized is

$$J = 50x^2(t_f) + \int_0^{t_f} (0.1u^2(t) + x^2(t))\,dt \tag{4-4}$$

where t_f is the specified terminal time. This performance measure tends to drive the final state $x(t_f)$ close to zero without wasting too much energy. The admissible values of the state and control variables are constrained by

$$-1.0 \le x(t) \le 1.0, \tag{4-5.1}$$

and

$$-5.0 \le u(t) \le 5.0 \cdot \tag{4-5.2}$$

Before the dynamic programming neural network algorithm can be applied, the system differential equation must be approximated by a corresponding difference equation, and the integral in the performance measure must be approximated by a summation. The system difference equation and the performance measure can be written as follows [10]

$$x(k+1) = (1-20\Delta t)\,x(k) + \Delta t \times u(k), \tag{4-6.1}$$

and

$$J = 50x^2(N) + \Delta t \sum_{k=0}^{N-1} (0.1u^2(k) + x^2(k+1)), \tag{4-6.2}$$

where Δt is small enough so that the control signal can be approximated by a piecewise-constant function that can change only at the time instants t = 0, Δt,....$(N\text{-}1)\Delta t$. N is the number of time instants from 0 to t_f and k is the index of time instants.

It is assumed in this problem that the state and control variables are allowed to take on only quantized values of the state set X={-1.0, -0.9,..., 0,..., 1.0} and the control set U={-5.0, -4.0,..., 5.0}. In addition, t_f will be specified as 1.0. Therefore, a 10x21 network

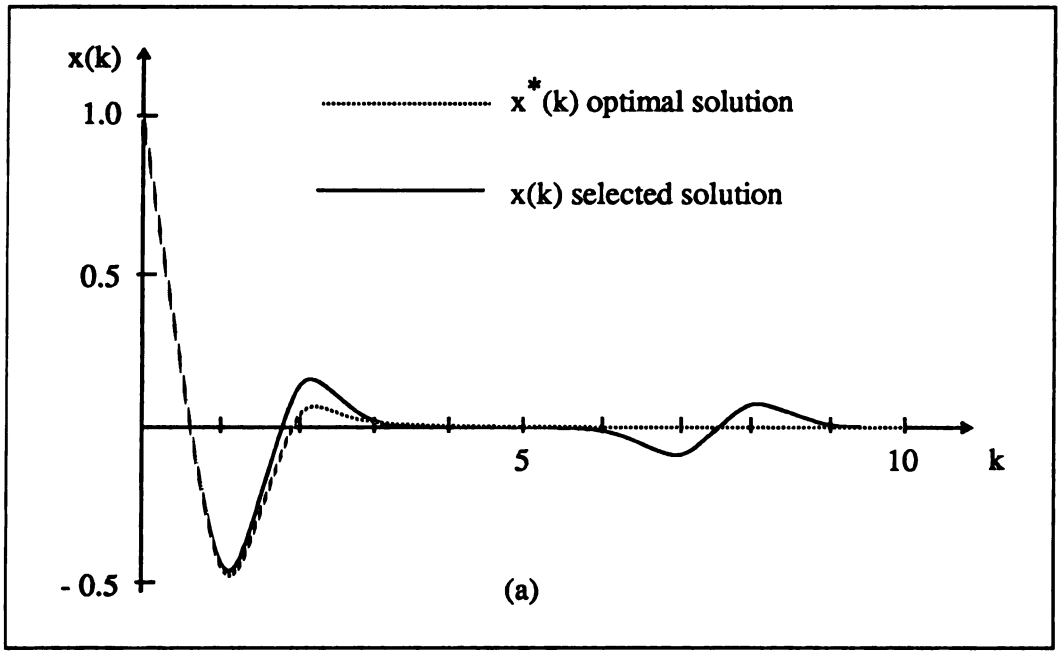
is needed to solve this problem. The kth stage of the network represents the time instant $k\Delta t$ and the processing elements correspond to the allowed values of state variables. Assume $x_i(k)$ is the state i of the kth stage time instant and $u_{ij}(k)$ is the control signal which drives $x_i(k)$ to $x_j(k+1)$. The cost of operation in the interval k to $k+1$ for specified quantized control $u_{ij}(k)$ is $J_{ki\,(k+1)j}$, where $0 \leq k \leq N-1$. It is equal to $0.1\Delta t u_{ij}^2(k) + x_j^2(k+1)$ if $x_j(k+1)$ belongs to the state set X. Otherwise, $J_{ki\,(k+1)j}$ will be set to a very high value to prevent the sub-path $x_i(k)$ to $x_j(k+1)$ from being selected. In addition, $J_{Ni\,(N+1)0}$ is equal to 50 times $x_i^2(N)$ which is the square of the difference between the final state and the zero state. Therefore, the connection weight $T_{xi,yj}$ is defined as

$$T_{xi,\,yj} = -\alpha\delta_{xy}(1-\delta_{ij}) - \alpha - \frac{\beta}{2}J_{xi,\,yj}(\delta_{(x+1)\,y} + \delta_{(x-1)\,y}) \qquad (4\text{-}7)$$

Figure 4-4a shows the optimal state trajectory and the state trajectory found by the proposed algorithm with initial state x(0)=1.0. The selected control signals $u(k)$ are equal to {5, -3, 2, 0, 0, 0, -1, 0, 1, 0}. These are very close to the optimal control signals $u^*(k)$= {5, -5, 0,....., 0}. Figure 4-4b depicts the optimal state trajectory and the selected state trajectory with initial state $x(0)$=-1. The optimal control signal sequence is {-5, 5, 0,...., 0} and the obtained control signal sequence is {-5, 3, -2, -1, 0, 0, 0, 1, 0, 0}. Again, these two state trajectories are very close.

4.3 Summary

Simulations of the network have been developed to demonstrate the robustness of the network. The neural network can provide a near-optimal solution in an elapsed time of only a few characteristic time constants of the circuit. This has been found to apply for DPPs as large as 64 stages with 64 states in each stage. The relationship between the quality of solutions and the parameters of the associated energy function was explored. Simulation shows that the minimum states can be preserved if the errors of modified connections is



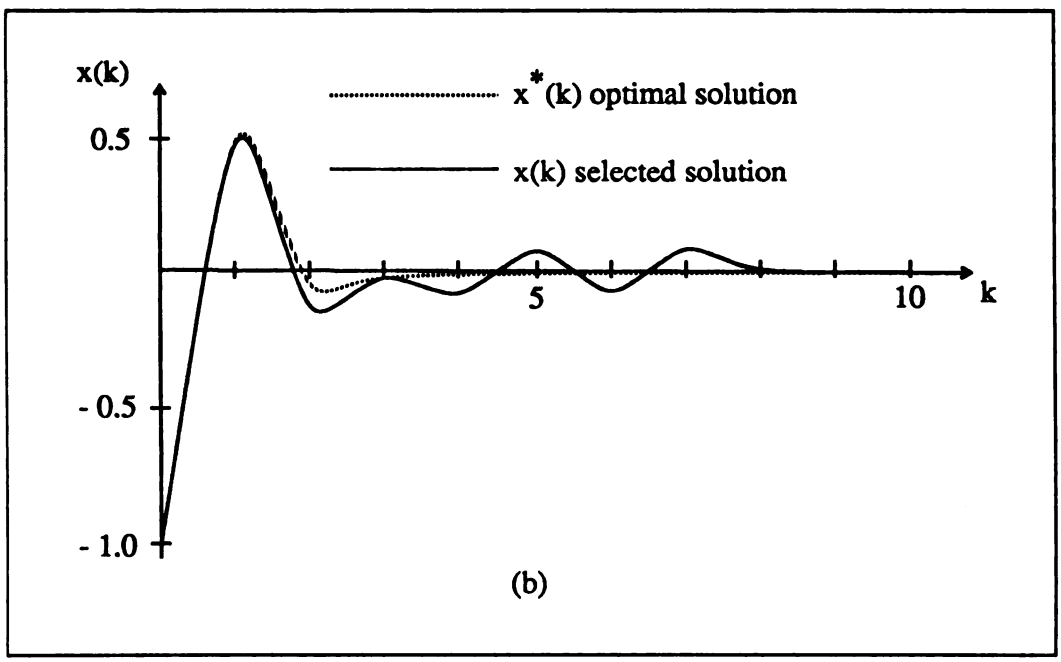


Figure 4-4. a) Optimal state trajectory and selected state trajectory for x(0)=1.0. b) Optimal state trajectory and selected state trajectory for x(0)=-1.0. The curves of state trajectories have been smoothed.

small. Moreover, the basins of attractions of the network's equilibria can be increased by appropriately increasing the parameters of the associated energy function. Finally, an application of the new network to an optimal control problem demonstrated the capability of the network.

Chapter 5

5.0 A Building Block Architecture and Problem Decomposition

In the first part of this chapter, an architecture for the dynamic programming neural network is presented. The architecture is based on a building block paradigm in which the network is constructed from neuron array and weight assignment chips. Because of its simple and regular structure, the architecture is a feasible implementation for dynamic programming neural networks with current VLSI technology. Moreover, this architecture can be further extended to other Hopfield-Tank type networks. Following the presentation of the architecture, a divide-and-conquer method for solving very large scale dynamic programming problems is described. A flight planning problem is used to illustrate the advantages and disadvantages of this method in terms of the network size, processing time, and quality of solutions.

5.1 A VLSI Building Block Architecture

Although artificial neural networks are composed of basic circuits emulating neurons and synapses, direct hardware implementation has been severely restricted in size by constraints of current VLSI technology. These constraints include real-estate considerations, and fan-in/fan-out and connectivity limitations. To alleviate these problems, especially in the case of moderate to large size networks, a building block paradigm in which a neural network is constructed by several VLSI chips implementing various components has been used for feed-forward networks [83]. Even though this paradigm may reduce the processing speed over that of single-chip designs due to off-chip routing, it allows the construction of networks with arbitrary feed-forward architectures by these pre-designed building blocks [83].

An architecture for dynamic programming neural networks is presented following this paradigm. For convenience, the network dynamic equation is written again.

$$C_{xi}\left(\frac{du_{xi}}{dt}\right) = -\frac{u_{xi}}{R_{xi}} - a\sum_{j\neq i} v_{xj} - a\sum_{y}\sum_{j} v_{yj} + an$$
$$-\frac{b}{2}\sum_{j}\left(d_{xi,(x+1)j}v_{(x+1)j} + d_{(x-1)j,xi}v_{(x-1)j}\right) \qquad (5\text{-}1.1)$$

where

$$\frac{1}{R_{xi}} = \frac{1}{\rho_{xi}} + \sum_{yj} T_{xi,yj}. \qquad (5\text{-}1.2)$$

Equation (5-1.1) can be expressed as

$$C_{xi}\left(\frac{du_{xi}}{dt}\right) = -\frac{u_{xi}}{R_{xi}} + f_{(x-1)i}(v_{x-1}) + f_{(x+1)i}(v_{x+1})$$
$$-a\sum_{j\neq i} v_{xj} - a\sum_{yj} v_{yj} + an \qquad (5\text{-}2.1)$$

where

$$v_x = [v_{x1}, v_{x2}, \ldots, v_{xm}]^T, \tag{5.2-2}$$

$$f_{(x-1)i}(v_{x-1}) = -\frac{b}{2}\sum_j (d_{(x-1)j,xi} v_{(x-1)j}), \tag{5-2.3}$$

$$f_{(x+1)i}(v_{x+1}) = -\frac{b}{2}\sum_j (d_{xi,(x+1)j} v_{(x+1)j}), \tag{5-2.4}$$

and m is the number of neurons in each stage. Let

$$w_{xi} = C_{xi}(\frac{du_{xi}}{dt}) + \frac{u_{xi}}{R_{xi}} + a\sum_{j \neq i} v_{xj} + a\sum_{yj} v_{yj} - an, \tag{5-3.1}$$

$$w_x = [w_{x1}, w_{x2}, \ldots, w_{xm}]^T, \tag{5-3.2}$$

$$F_x = [f_{x1}(v_x), f_{x2}(v_x), \ldots, f_{xm}(v_x)]^T, \tag{5-3.3}$$

and

$$D_{(x-1)x} = \frac{b}{2}\left[c_{ij}\right]_{i,j=1}^{m}, \text{ where } c_{ij} = d_{(x-1)i,xj}. \tag{5-3.4}$$

From equation (5-1.1), it can be shown that

$$w_x = F_{x-1} + F_{x+1} = D_{(x-1)x}^T \bullet (-v_{x-1}) + D_{(x+1)x}^T \bullet (-v_{x+1}). \tag{5-4}$$

Therefore, the network can be decomposed into an element (chip) implementing the w_x's and connection chips implementing the $D_{(x-1)x}$'s. Figure 5-1 shows part of such a

decomposed architecture for the dynamic programming neural network. Consider an $n \times m$ network with n neuron array chips and $2n$-2 connection chips. Each neuron array chip, shown in Figure 5-2a, has m processing elements, shown in Figure 5-2b, and each connection chip, shown in Figure 5-3, consists of $m \times m$ matrix-like connections. Notice that every neuron array chip has connections to its two neighbor chips, a global input $V_{all} = \sum_{yj} v_{yj}$, and an output $V_{sub}(x) = \sum_{j} v_{yj}$. V_{all} is the sum of all neuron outputs and can be implemented by an analog adder shown in Figure 5-1. $V_{sub}(x)$ is the sum of neuron outputs in stage x. Notice that the network connectivity of this architecture is $O(nm^2)$ instead of $O(n^2m^2)$, which is generally required for a Hopfield-type network with $n \times m$ totally connected neurons.

The resistance elements of the connection chips and the input capacitances of the neuron array chips may be designed as adaptable circuits to allow the circuit to be used to solve different problems. Many methods have been developed for adaptable capacitors and resistors, including implementations of connection weights and input capacitances with digital and analog programmable circuits [83-86]. Eberhardt, *et al.*, have designed a 32 $\times$ 32 connection chip which utilized the four-quadrant Gilbert multiplier to convert charge stored on capacitors into conductances for feed-forward neural networks. These designs can be utilized to implement this network.

If a host computer, an interface between the host and the network, and an A/D converter are considered along with the dynamic programming neural network, then this integrated system is capable of solving different problems. This system, shown in Figure 5-4, works as follows. Initially, a problem is specified in the host computer. Then, the computer will calculate the connection weights, download the weight values to connection chips, and set up initial conditions via the interface. To download a weight value to a resistance element and an initial value to an input capacitance, each adaptive resistance and capacitance must be addressable. With the associated address selected by the host

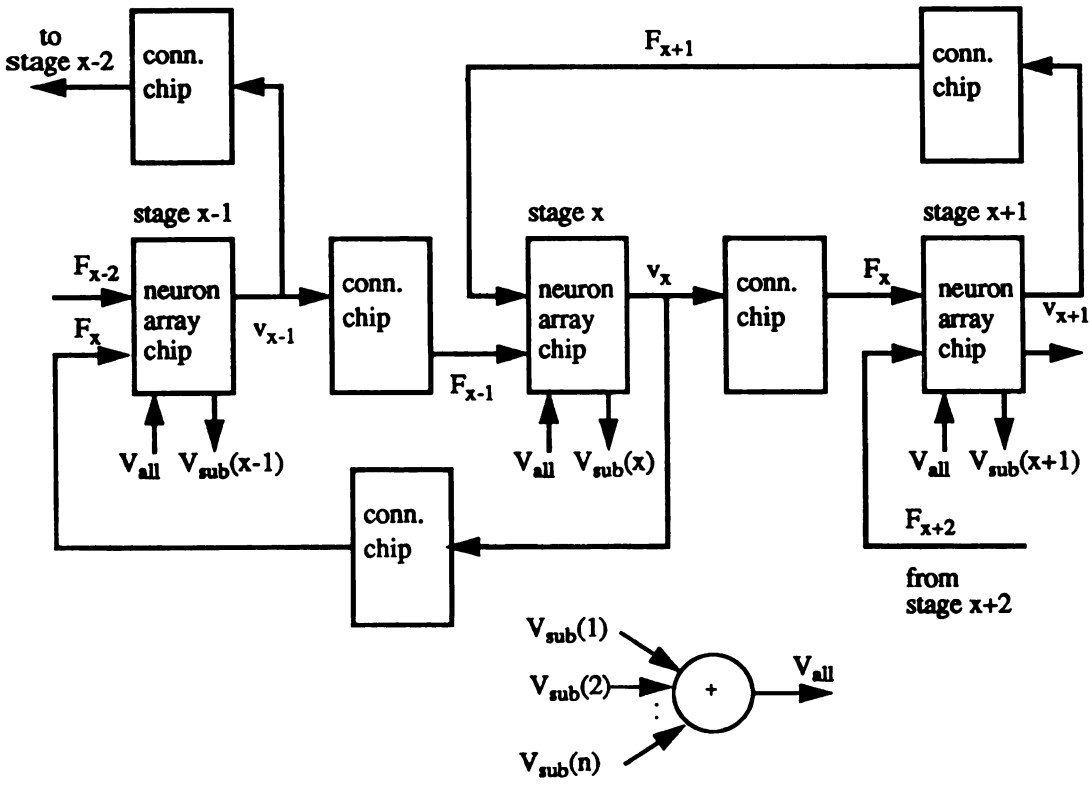
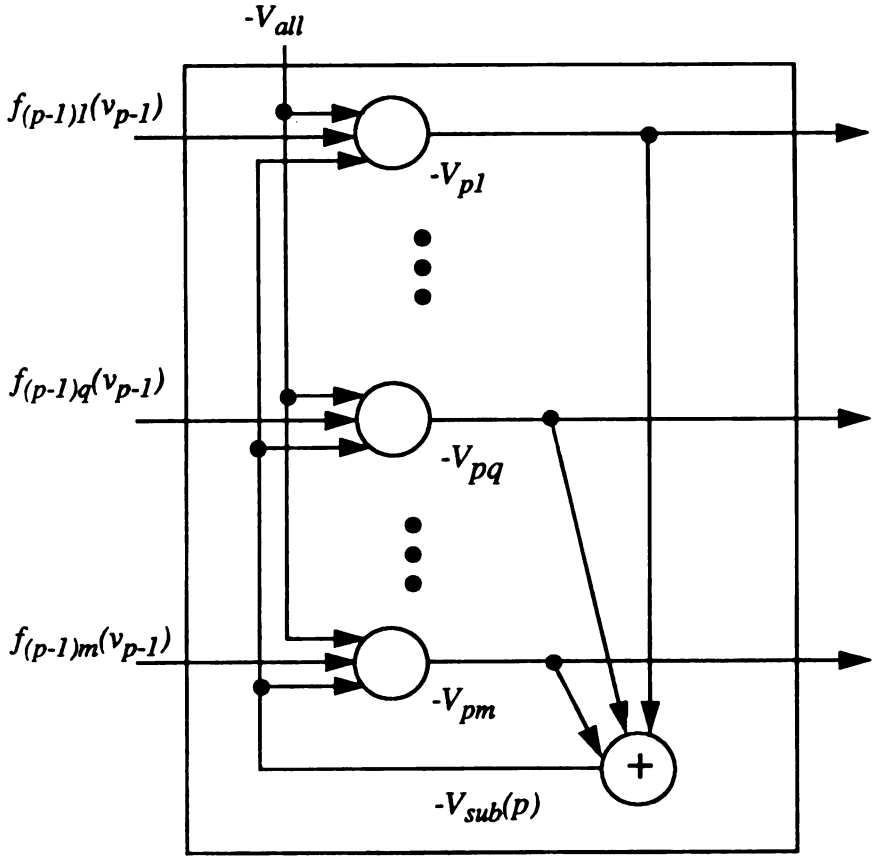
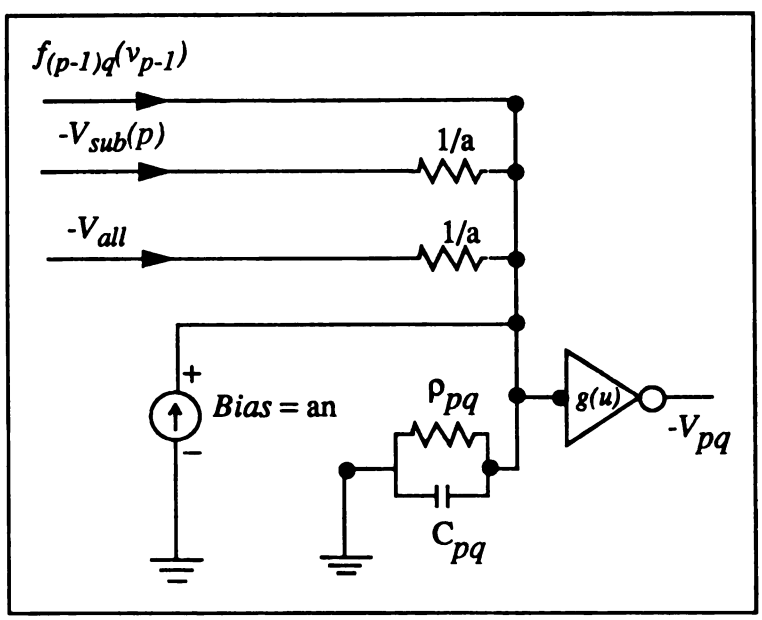


Figure 5-1. A building block architecture (only 3 stages shown) for the dynamic programming neural network.



(a)



(b)

Figure 5-2. a) A neuron array. b) A neuron design.

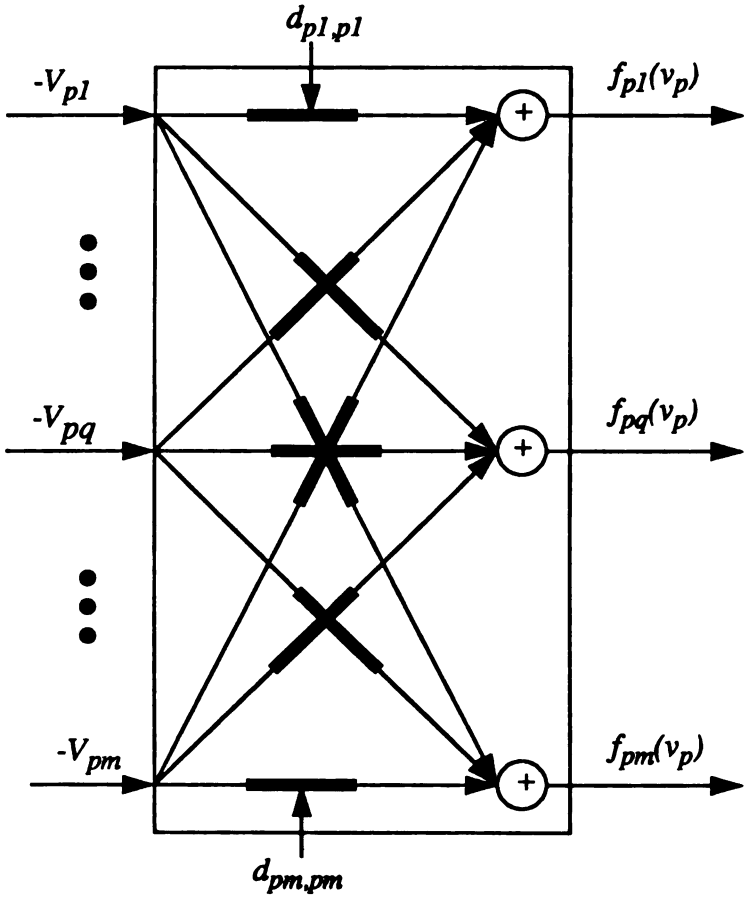


Figure 5-3. A connection matrix.

computer, each connection and initial condition can be "programmed" by downloading the corresponding values which are converted into analog signals by the interface. The host computer then starts the dynamic programming neural network and samples each processing element output at a certain rate to check if the network reaches a steady state. The steady state can be read via the A/D converter after the network has converged.

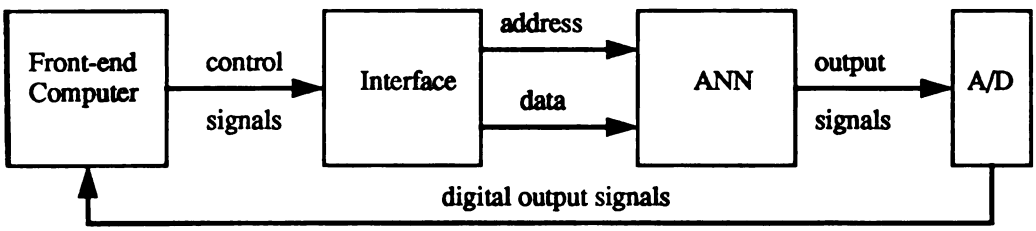


Figure 5-4. A neuronprocessing system for dynamic programming.

5.2 Problem Decomposition

The implementation of large scale neural networks has been severely restricted by current technology. One way to cope with this problem is to use a divide-and-conquer strategy. The advantage is that many smaller and feasible networks can be used to solve larger problems. However, the time to find a solution will increase accordingly, and the solution quality may not be as good as that for the solutions found without decomposition.

5.2.1 Algorithm

This decomposition method is described as follows.

(1) Decompose the problem to be solved into sub-problems with size less than or equal to that of the available smaller network. Figure 5-5a depicts a decomposition which divides a 4×4 problem into 4 2×2 sub-problems.

With each sub-problem representing a state node, a new problem is formed which is shown in Figure 5-5b using the same example. The new metric distances between two nodes in two adjacent stages will be assigned by a predefined function of the metric distances of the original problem.

(2) Find a near-optimal solution for the new problem.

(3) Find a near-optimal solution for every sub-problem which corresponds to a state visited by the near-optimal path of the new problem.

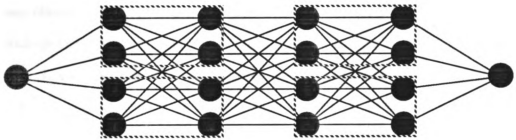
(4) Link the near-optimal solutions found in step (3) as a complete solution.

5.2.2 Examples

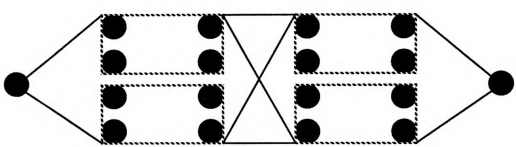
The problem to be solved is to find a minimum cost path from a start point to an end point in a map consisting of 50 x60 points. Each point has an associated cost. The set of 50 x60 cost values is given in Appendix A. The total cost of the path is the sum of the costs of the points along the path. To reduce the network size, a divide-and-conquer strategy is used. This method divides the 50x60 map into 100, 5x6 regions. Then, with each region considered as a point, the 50x60 map can be reduced to a 10 x10 map. The cost of each point of the reduced 10x10 map is simply the sum of the costs of the 30 points in the corresponding 5x6 region. The set of cost values for the reduced 10x10 map is given in Appendix B. A minimum cost path in the reduced map then can be obtained using a smaller network. Finally, a minimum cost path in the original 50x60 map can be generated by finding each sub-path in each corresponding 5x6 region which is visited in the minimum cost path of the 10x10 map.

To convert the problem to a neural network, the number of stages of the network and the states in each stage must be determined first. Assume the desired path is regular and smooth, the number of stages for a start point (s_x, s_y) and an end point (e_x, e_y) can be set to

$$M = c \times max\left(|e_x - s_x|, |e_y - s_y|\right)$$



(a)



(b)

Figure 5-5. a) A 4×4 problem is divided into 4 2×2 sub-problems.
b) A new 2×2 problem with each state representing a 2×2 sub-problem.

where c > 1. The larger the coefficient c is, the more potential solutions the network will have. However, the network size will increase. In simulations, c is chosen as $1.5 \le c \le 2.0$ which gives a reasonable trade-off between the solution quality and network size.

Let *T(i)* be the set of states in the *i*th stage. Assume the path directions are only allowed in the -45, 0, and 45 degree directions. The possible states in the *i*th stage are derived in two steps. The first step where $0 \le stage\,(i) \le \left\lceil \frac{M}{2} \right\rceil$ begins from the start point to the half-way point of the path as follows. Let *T(0)* be the set of the starting state (s_x, s_y). The state set of the first stage is defined as

$$T(1) = \{t \mid (|s_x - t_x| \le 1,\ |s_y - t_y| \le 1)\ and\ (|e_x - t_x| \le M\ \ (|e_y - t_y| \le M))\}.$$

The states in *T(1)* are those which are the neighbor states of (s_x, s_y) and whose distances along x and y axis to (e_x, e_y) are less than the number of stages *M*. Define $dir\,(p, q)$ as the direction from p to q, where $p \in T(i-2)$ and $q \in T(i-1)$. Then each state t in *T(i)* where i > 1 must satisfy the following four conditions:

1. $|q_x - t_x| \le 1,\ |q_y - t_y| \le 1$
2. $t \ne q$
3. $|e_x - t_x| \le M-1,\ |e_y - t_y| \le M-1$
4. $dir\,(p, q) - dir\,(q, t) = -45,\ 0,\ or\ 45\ degrees.$

The second step, $\left\lceil \frac{M}{2} \right\rceil < i \le M$, starting from the end point to the half-way point of the path in backward fashion, follows the same procedure as the first step.

Figure 5-6 depicts parts of the network construction procedure in a 10x10 map. The starting point is (1,1) and the ending point is (9,9). Suppose *M*=14, then the first step goes from stage 1 to stage 7, and the second step goes from stage 14 to stage 8. As can be seen in Figure 5-6, the states in the first stage are (0,0), (0,1), (0,2), (1,0), (1,2), (2,0), (2,1), and

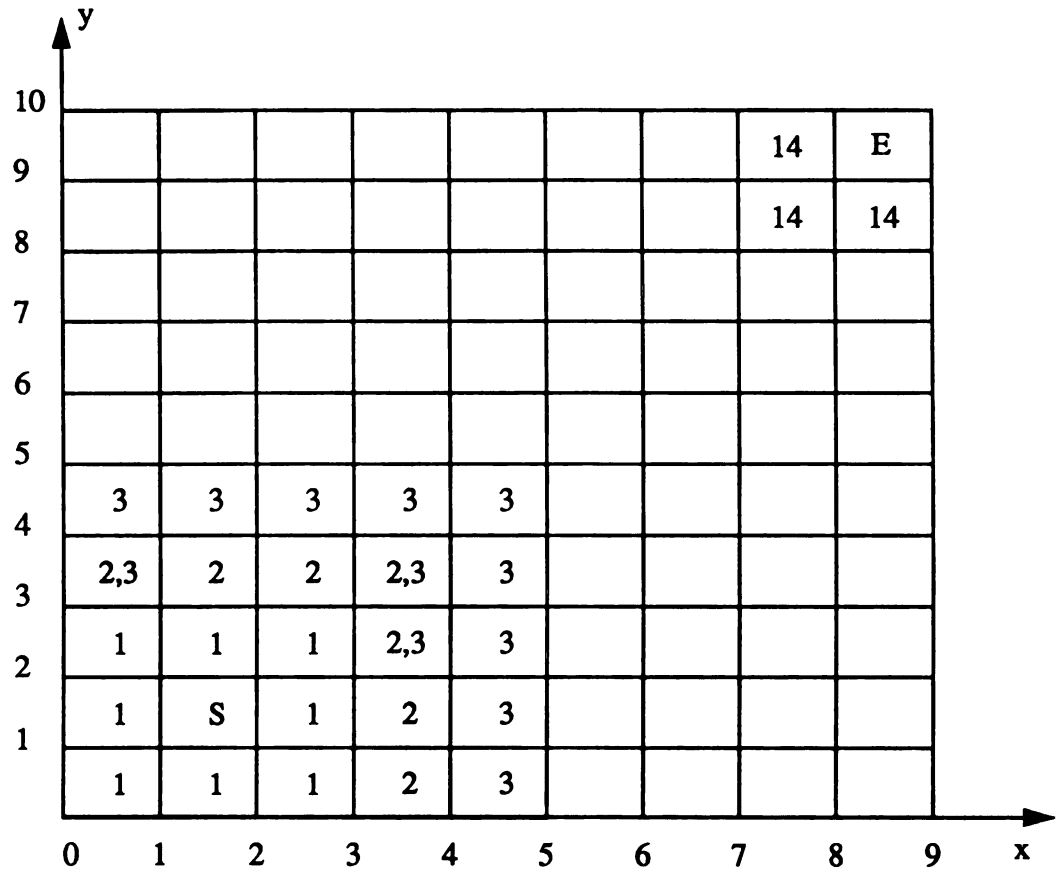


Figure 5-6. An example illustrating how to determine the states for each stage.

(2,2). The states in the second state are (0,3), (1,3),..., (3,1), and (3,0). The final stage, i.e., stage 14, consists of states (9,8), (9,9), (8,9), and (8,8).

After the stage number and the number of states in each stage have been determined, the connection weights $d(u,v)$ where $u \in T(i)$ and $v \in T(i+1)$ need to be selected to build a network. There are three cases:

Case 1: $u = v$.

Then $d(u, v) = 0$.

Case 2: $|u_x - v_x| \leq 1$ and $|u_y - v_y| \leq 1$ but $u \neq v$.

$d(u, v) = f(cost(v))$ if $f(cost(v)) \leq threshold$, else $d(u, v) = D$, where $f(\cdot)$ is a function which normalizes *cost (v)* such that the function values fall in [0, 10], threshold is an adjustable positive value in [5, 10] and D is set to a large value (over 100).

Case 3: $|u_x - v_x| > 1$ or $|u_y - v_y| > 1$.

To prevent an illegal sub-path from being selected, the connection weight is set to a large value as

$$d(u, v) = max(threshold \times (|u_x - v_x| + |u_y - v_y|), D).$$

5.2.3 Simulation

In the first example starting from (7, 15) and ending at (37, 21) in the original 50x60 map (from point (1,2) to point (7, 3) in the reduced 10x10 map), 8 stages are used for the network and the average number of states in each stage is about 15. The path shown in Figure 5-7, with a cost 16.83, is obtained by the method with decomposition. Figure 5-7 also shows the path found by the method without decomposition, with a cost 15.54, and the optimal path. Figure 5-8 shows the path with a cost 11.96 from point (2, 3) to point (28, 39) in the original 50x60 map (from point (0, 0) to point (5, 6) in the reduced 10x10 map) for the second example. The path, also shown in Figure 5-8, selected by the method without

decomposition has a cost of 12.21. Table 5-1 is a comparison of solution quality, processing time, and the network size required for the methods with and without decomposition in the first example. With the time increment step set to 0.01 in the simulation, the networks for the problems with the 5x6, 10x10, and 50x60 maps approximately require 200, 300, and 1,000 iterations to converge, respectively. Therefore, the number of iterations required for the method with decomposition in the first example is calculated as $8 \times 200 + 300 = 1,900$. With decomposition, the network size required is about 8x15 for the first example. However, the network is as large as 60x200 if the problem is not decomposed.

5.3 Summary

A building block based neural network architecture for dynamic programming was presented. The advantage of this proposed architecture is that the networks can be implemented with some simple well-designed VLSI chips and a reduced network connectivity. This architecture can also be extended to some other Hopfield-type networks. A problem decomposition method was described with a large-scale flight planning problem as an example.

Method	Cost	Processing iterations	Network size
With decomposition	16.83	~1,900	~8x15
Without decomposition	15.54	~1,000	~60x200

Table 5-1. Comparisons of solution quality, processing time, and the network size required for the methods with and without decomposition for the first example.

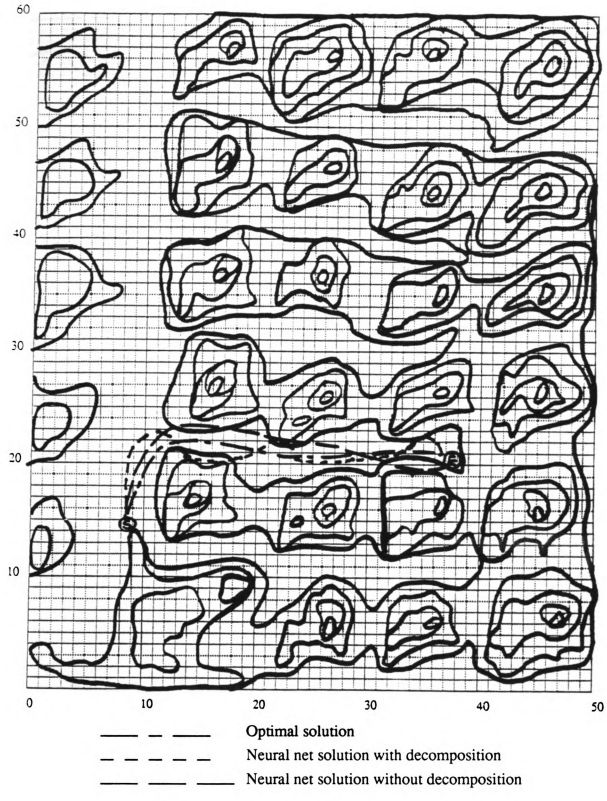


Figure 5-7. Different solutions for the first example.

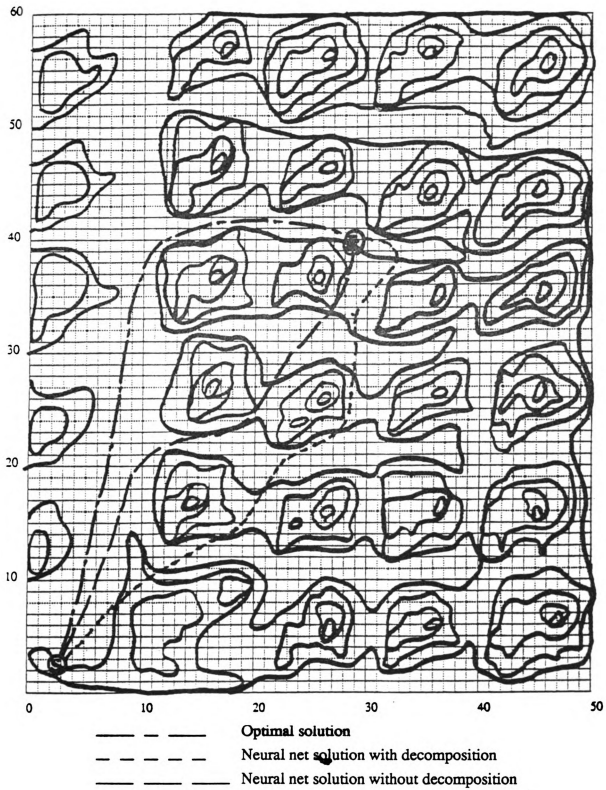


Figure 5-8. Different solutions for the second example.

Chapter 6

6.0 Application: Incremental Autoassociative Memory

For an autoassociative memory, each desired memory pattern can be considered as a state (path) with minimum energy (cost). Thus, the autoassociative memory problem can be viewed as a problem of finding the minimum energy state in a constrained search space dependent on the initial condition. The application, based on dynamic programming techniques used to find minimum cost paths, is very simple and straightforward in the sense that memory patterns can be easily added to an existing associative memory without any modification to the previous connections. Analysis reveals that each stored memory pattern corresponds to a global minimum state of the formulated network. Simulation also shows that every desired pattern can be successfully stored and recalled and the extent of the basins of attraction of the global minima can be adjusted by selecting the parameters of the network energy function. Two examples, with the first one designed to correct single-error codes and the second to distinguish three images, are used to demonstrate the power of the formulated network. The issue of fault tolerance of the autoassociative memory network is also considered. The network can recall the stored memory patterns very successfully even when noise is added to the inputs. Moreover, the network can detect, locate, and recover from a single stuck-at fault. Comparison with two other networks from the literature shows that the incremental autoassociative memory network is more flexible and robust.

6.1 Introduction

Suppose that a set of memory patterns $\{Y^1, Y^2, ..., Y^l\}$ is to be stored, where $Y^r \in R^n$ for $r = 1$ to l. Given a stimulus $X = Y^r + \varepsilon$ where ε is sufficiently small, an autoassociative memory is designed in such a way that the output pattern Y^r can be recalled successfully. In other words, an autoassociative memory is designed to store a set of desired memory patterns where each pattern is attractive in the sense that any input pattern in its region of attraction will eventually converge to it.

The Hopfield-Tank model is capable of implementing associative networks [14]. However, some of the proposed networks do not guarantee that each desired memory pattern can be stored [14,38,39]. Moreover, some of them suffer from rather complex synthesis procedures such that, in order to store large memory patterns, the network connection weights must be determined by solving a large number of equations[41]. Additionally, a major drawback against them is that the whole network must be reconstructed if additional memory patterns are to be stored.

It has been noted that the associative memory problem can be considered as an optimization problem [7]. For example, the problem of recognizing a familiar face can be interpreted as finding which one of those acquainted faces in the memory most closely resembles the test face. That is, to find which one maximizes the overlap between the test face and the acquainted faces.

Dynamic programming, one of the techniques used to solve combinatorial optimization problems, can be used to find optimal paths from source to destination. If each desired memory pattern is considered as a shortest path, then the associative memory problem can be regarded as a dynamic programming problem, i.e., finding the shortest path within a constrained search space dependent on the initial condition.

A coding procedure for autoassociative memory based on dynamic programming neural networks will be presented. It will be shown that each desired memory pattern can

be successfully stored in the network and each memory pattern corresponds to a global minimum state of the associated energy function. Furthermore, the extent of the basins of attraction of the stored memory pattern can be controlled by selecting the parameters of the associated energy function.

One of most significant advantages of this autoassociative memory network is the inherent fault tolerance. The fault models considered in this chapter are noisy inputs and single stuck-at faults. The network can recall the stored memory patterns in a very noisy environment. Moreover, the network has the ability to detect, locate, and recover from a single stuck-at fault.

This chapter explores the autoassociative network as follows. Section 6.2 presents the network formulation procedure and analysis. Simulations and examples are given in Section 6.3. The issue of fault tolerance is addressed in Section 6.4 and Section 6.5 compares the proposed network to two other known networks.

6.2 Autoassociative Memory Network

6.2.1 Formulation

Assume that a set of memory patterns $Y = \{Y^1, Y^2, ..., Y^l\}$ is to be stored in an autoassociative memory, where $Y^r = [y_1^r, y_2^r, ..., y_n^r] \in R^n$ is in binary form, i.e., y_i^r is either 1 or 0. To design an autoassociative memory using a dynamic programming network, the minimum paths corresponding to the desired memory patterns must be defined.

Assume a dynamic programming network with n stages and $2l$ states for each stage. Let Path(r) be the minimum path associated with memory pattern Y^r, where Path(r) starts from the source node, passes through states $p_1^r, p_2^r, ...,$ and p_n^r in stage 1,2...,n, respectively, and finally ends at the destination node. The distance between p_x^r and p_{x+1}^r is defined as $d(x, p_x^r, p_{x+1}^r)$. Let p_0^r be the source node, p_{n+1}^r be the destination node, and

$s(x, i)$ be the ith state in stage x. We define $p_x^r = s(x, r + y_x^r \times l)$, and $d(x, p_x^r, p_{x+1}^r) = 0$, where $0 \le x \le n$ and $1 \le r \le l$. The other distances, between any two states in two adjacent stages, are all set to a large positive number D. Therefore, the total distance of each minimum path is zero and the others are at least D.

Figure 6-1 shows the network which can store three memory patterns. It will be shown later that the designed network has exactly l global minimum states which correspond to the desired memory patterns in a one-to-one fashion.

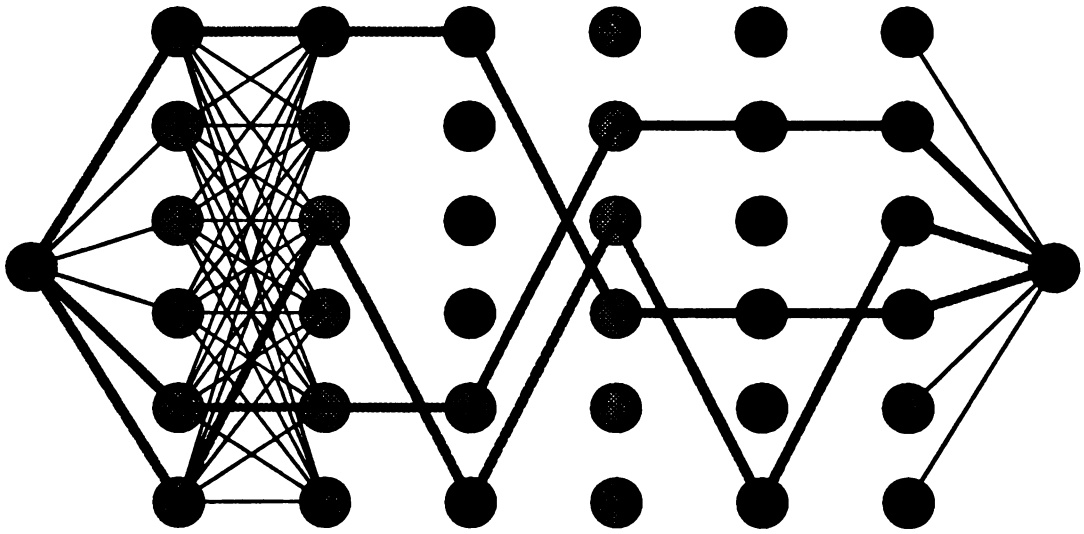


Figure 6-1. A 6 by 6 network (full connection not shown) which can store patterns (0,0,0,1,1,1), (1,1,1,0,0,0) and (1,0,1,0,1,0). Three shortest paths are shown in thick lines.

6.2.2 Definitions

The following definitions will be used in the remainder of this chapter.

i) The *domain* and *range* of the sigmoid function g are defined as $U=R^N$ *and* $V=[0, 1]^N$,

respectively, where $N = m \times n$. Every vector $v \in V$ can be expressed as $[v_{11}, v_{12}, \ldots, v_{mn}]^T$, where $0 \leq v_{xi} \leq 1$. The zero vector of V is denoted as 0_N.

ii) A vector $c \in V$ is said to be a *valid corner state* if c_{xi} is either unity or zero and

$$\sum_i c_{xi} = 1 \text{ for } x=1 \text{ to } n.$$

iii) The *valid corner set* C is defined as the set of all valid corner states.

iv) The *path vector* T^r *associated with memory pattern* Y^r is defined as

$$T^r = [t_1^r, t_2^r, \ldots, t_n^r], \text{ where } t_x^r = r + 2l \times y_x^r, \text{ and } 1 \leq x \leq n.$$

v) The *minimum state* V_{min}^r *associated with memory pattern* Y^r is defined as

$$V_{min}^r = [v_{11}^r, v_{12}^r, \ldots, v_{mn}^r], \text{ where } v_{xi}^r = 1 \text{ if } i = t_x^r, \text{ and 0 otherwise, } 1 \leq x \leq n,$$

$1 \leq i \leq 2l$, and $1 \leq r \leq l$. Thus V_{min}^r is a valid corner state.

6.2.3 Analysis

The following propositions provide a theoretical insight into the autoassociative memory network. For completeness, some of the relevant propositions which were presented in Chapter 3 are restated here without proof.

Proposition 1. v is a valid corner state if and only if $E1(v) = 0$.

Proposition 2. Every V_{min}^r is a global minimum state.

Proof: Since V_{min}^r is a valid corner state, $E1\,(V_{min}^r) = 0$. Furthermore, we have

$$E2\,(V_{min}^r) = \sum_x \sum_i \sum_j d_{xi,(x+1)j} v_{xi}^r v_{(x+1)j}^r + d_{(x-1)j,xi} v_{xi}^r v_{(x-1)j}^r$$
$$= \sum_x \sum_i d\,(x, p_x^r, p_{x+1}^r) = 0 \cdot$$

Therefore, it follows that

$$E(V^r_{min}) = \frac{a}{2}E2(V^r_{min}) + \frac{b}{4}E2(V^r_{min}) = 0 \cdot$$

Since $E(v) \geq 0$ for every $v \in V$, V^r_{min} is a global minimum state. Q.E.D.

Proposition 3. There are exactly l global minimum states.

Proof: Suppose there is another global minimum state v which does not correspond to any of the minimum paths. Since v is a global minimum state, E2(v) is equal to zero. Consequently, v must be the zero state 0_N. However, 0_N is not a valid corner state. Therefore, according to Proposition 1, v is not a global minimum state. Q.E.D.

Some undesired local minimum states, which correspond to paths that are either invalid or with a total distance greater than or equal to D, may also exist in the network. However, the following proposition states that they are all on the boundary of the unit hypercube. This gives a guide as to how to set the initial conditions as described in the next section.

Proposition 4. Every local minimum state of E is on the boundary of V.

6.3 Examples

For convenience, rewrite the simplified network equation as

$$C_{xi}\left(\frac{du_{xi}}{dt}\right) = -\alpha\sum_{j\neq i} v_{xj} - \alpha\sum_{y}\sum_{j} v_{yj} + \alpha n$$
$$-\frac{\beta}{2}\sum_{j}\left(d_{xi,(x+1)j}v_{(x+1)j} + d_{(x-1)j,xi}v_{(x-1)j}\right) \cdot$$

6.3.1 Initial Conditions

The initial condition Z^r associated with memory pattern Y^r for the input pattern $X = [x_1, x_2, ..., x_n]$ is defined as $Z^r = [z^r_{11}, z^r_{12}, ..., z^r_{mn}] \in R^N$ where $z^r_{ki} = h$ if $i = r + x_k \times l$, and 1 - h otherwise. h is usually selected in the range (0, 0.2) depending on the application [8]. Since Proposition 4 shows that the undesired local minimum states lie on the boundary of V, the initial conditions should not be set too close to the boundary to keep them from falling into the basins of attraction of the undesired local minima.

Given two memory patterns [0, 0, 0, 0, 0, 0, 0, 0] (code A) and [1, 1, 1, 1, 1, 1, 1, 1] (code B), a network with $\alpha = 50$ and $\beta = 4$ has been designed. Table 6-1 shows the number of attracted patterns to global minimum states and local minimum states of the network with different initial conditions, where H is the Hamming distance between the stored patterns. As can be seen in Table 6-1, the number of attracted patterns increases when h is increased. The case of $h = 0.20$ gives the largest number (184) of attracted patterns; however, classification errors also occur since 41 patterns with Hamming distances5 or more units away from the stored patterns are attracted.

Number of attracted patterns	Global minima		Local minima
	$0 \le H \le 4$	$5 \le H \le 6$	
h = 0.01	91	0	165
h = 0.05	107	0	149
h = 0.10	107	0	149
h = 0.15	131	0	125
h = 0.20	184	41	31

Table 6-1. Number of attracted patterns for global and local minima with different initial conditions.

6.3.2 Simulation

Three major steps are followed to verify the above by simulation.

Step 1.0 The memory patterns to be stored and the test pattern X are specified.

Step 2.0 The local distance $d_{xi,(x+1)j}$ is generated according to Section 6.2.1.

Step 3.0 Set $r = 1$;

While ($r \leq l$)

3.1 Set the initial condition associated with memory pattern Y^r;

3.2 Relax the network;

3.3 Check the network for convergence;

3.4 If the converged state corresponds to memory pattern Y^r

then stop;

Else increase r by 1;

3.5 Reject the test pattern X and stop;

6.3.3 Basins of Attraction

To understand how the parameters a and b affect the basins of attraction of the global minimum states, the following experiment has been done using the previous example in Section 6.3.1. With h and a fixed to 0.10 and 50, respectively, β is changed from 1 to 5. For every value of β, $2^8 = 256$ test patterns have been fed to the network. The numbers of attracted patterns of different Hamming distances to code A and B for each value of β is listed in Table 6-2. It can be seen that the number of attracted patterns, which may be used as an index of the basin of attraction, increases when β is increased. However, classification errors occur as β becomes greater than 4. It also can be seen, interestingly, that the network perfectly recovers the codes with H = 2 in the case of $\beta = 5$.

β \ H / code		0	1	2	3	4	5
1	A	1	0	0	0	0	0
	B	1	0	0	0	0	0
2	A	1	0	0	0	0	0
	B	1	0	0	0	0	0
3	A	1	8	0	0	0	0
	B	1	8	0	0	0	0
4	A	1	8	22	18	3	0
	B	1	8	22	20	4	0
5	A	1	8	28	39	24	2
	B	1	8	28	49	41	7

Table 6-2. Number of attracted patterns for global minima with different values of β.

6.3.4 Examples

With $\alpha = 15$, $\beta = 2$, and $h = 0.05$, the first example shows that a network can be designed to correct single-error codes. Eight six-bit code words shown in Figure 6-2a with three message bits and three check bits have been stored in a 16×6 network. The code words of Figure 6.2a are plotted on the 6-cube map of Figure 6.2b. Each code word is indicated by the corresponding letter and all cells one distance away from a code word are marked with an $\times$. As can be seen in Figure 6.2b, each code word is surrounded by 6 single-error codes. Simulations show that the network converges to the correct code word when each of the 6 single-error codes is fed to the network.

The second example demonstrates the power of the artificial neural networks in distinguishing among three patterns, an English letter, a Chinese character, and a Greek

Code	M1	M2	M3	C1	C2	C3
a	0	0	0	0	0	0
b	0	0	1	1	1	0
c	0	1	0	0	1	1
d	0	1	1	1	0	1
e	1	0	0	1	0	1
f	1	0	1	0	1	1
g	1	1	0	1	1	0
h	1	1	1	0	0	0

$C1 = M1 \oplus M3$

$C2 = M2 \oplus M3$

$C3 = M1 \oplus M2$

(a)

M1 M2 = 00

C2 C3 \ M3 C1	00	01	11	10
00	a	a'	b'	a'
01	a'	e'	d'	
11	c'		b'	f'
10	a'	b'	b	b'

M1 M2 = 01

C2 C3 \ M3 C1	00	01	11	10
00	a'		d'	h'
01	c'	d'	d	d'
11	c	c'	d'	c'
10	c'	g'	b'	

M1 M2 = 10

C2 C3 \ M3 C1	00	01	11	10
00	a'	e'		h'
01	e'	e	e'	f'
11	f'	e'	f'	f
10		g'	b'	f'

M1 M2 = 11

C2 C3 \ M3 C1	00	01	11	10
00	h'	g'	h'	h
01		e'	d'	h'
11	c'	g'		f'
10	g'	g	g'	h'

(b)

Figure 6-2. a) Eight single-error-correcting code words composed of 3 message bits and 3 check bits and parity check table. b) 6-cube map for the code of (a).

letter, as shown in Figure 6.3a. The network with parameters $\alpha = 15$, $\beta = 2.5$, and $h = 0.10$ has been tested with hundreds of patterns with randomly introduced noise. The noise is added to the patterns in such a way that every pixel may be reversed as determined by a probability p. Table 6-3 shows the success rate of recall with p being varied from 5% to 33%. 100 test patterns with random noise have been fed to the network for each stored pattern and for each p. It can be seen that the network performs well when $p \leq 20\%$. Figure 6.3b shows some of the test patterns. Furthermore, three test patterns with part of the images opaque, as shown in Figure 6.3c, have been also successfully recalled.

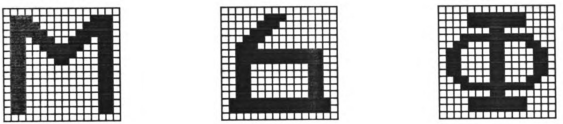
For a third example, the same procedure has also been run through with airplane, tank, and helicopter patterns as used in [39]. The simulation results are also listed in Table 6-3. Notice that the results in both examples are very close.

p%	5%	10%	15%	20%	25%	30%
Figure 5 examples	100%	100%	98.5%	98%	88%	70.5%
Examples from [39]	100%	100%	97.7%	97%	86%	65.7%

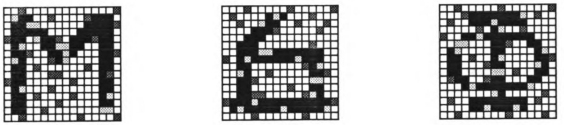
Table 6-3. Success rate of recall for the second and third example with different percentages of pixels being incorrect.

6.4 Fault Tolerance in the Autoassociative Memory Network

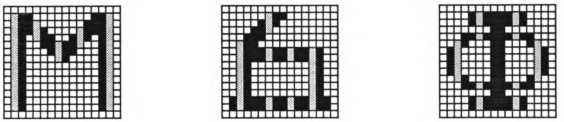
Unlike conventional computers, stored information in a neural network is determined by its structure, processing elements, and connections. Since the output results are collective decisions of all parameters together, the loss of information due to faults, such as damaged connections and malfunctioning processing elements, will still often result in the network obtaining a correct output. This provides an inherent fault tolerance property for neural networks. However, the fault tolerance works only when the neural



(a)



(b)



(c)

Figure 6-3. a) Three stored patterns. b) Three test patterns with random noise. c) Three test patterns with opaque image.

network is large enough to store the pieces of information in a redundant way [87]. Moreover, if faults occur in some important neurons, the fault tolerance ability may be degraded. For example, if stuck-at faults occur in some of the output neurons in a feed-forward network, the results will always be incorrect. If other neurons are faulty, the learning process may be slowed or divergent, and stored information may be lost.

Fault models to be examined in this chapter include noisy inputs and single stuck-at-faults. Neural networks are expected to operate in noisy environments, where the received inputs do not exactly match the desired inputs, such as in pattern recognition problems where only partial information is provided as the input. This partial information can be viewed as noisy input. In the binary domain, a noisy input is considered as a vector with some bits complemented. For such noisy input, the results in Table 6-3 show the success rate of recall with p being varied from 5% to 33%. It can be seen that the network can recall very successfully when $p \leq 20\%$.

Let *neuron(x, i)* denote the ith neuron of stage x. *Neuron(x,i)* is said to be stuck at 1 or 0 if v_{xi} is equal to 1 or 0 for all t (time), respectively. For a single stuck-at fault, only two cases, stuck-at-1 and stuck-at-0, are considered. From the definitions in Section 6.2.2, the neuron in the xth stage of Path(r), which corresponds to the memory pattern Y^r, is $neuron\,(x, r+2l\times y_x^r)$. For the first case, assume *neuron(x,i)* is stuck at 1. If $i \neq r+2l\times y_x^r$, then the network will not converge when the memory pattern Y^r is applied, since the output of $neuron\,(x, r+2l\times y_x^r)$ will be forced to 0. This is because *neuron(x, i)* is stuck at 1 and one neuron's output can, by definition, be one. On the other hand, if $i = r+2l\times y_x^r$, i.e., $neuron\,(x, r+2l\times y_x^r)$ is stuck at 1, then the memory pattern Y^r can be recalled when it is applied. For the second case, assume *neuron(x, i)* is stuck at 0. If $i = r+2l\times y_x^r$, i.e., $neuron\,(x, r+2l\times y_x^r)$ is stuck at 0, then the memory pattern Y^r will not be recalled when it is applied since the output of $neuron\,(x, r+2l\times y_x^r)$ is always 0. However, if $i \neq r+2l\times y_x^r$, the memory pattern Y^r will be recalled when it is applied since the network's convergence is not affected by the faulty neuron.

To locate a stuck-at neuron, it is necessary to apply every memory pattern to the network. Four cases can occur according to the stuck-at-fault analysis derived in the previous paragraph.

Case 1: The network converges only when the memory pattern Y^r is applied.

There is a stuck-at-1 fault in one of the neurons in Path(r). Moreover, the divergent neurons must be in the same stage, say x, when memory patterns other that Y^r are applied. Then $neuron\,(x, r + 2l \times y_x^r)$ is the faulty neuron.

Case 2: The network diverges only when Y^r is applied.

There is a stuck-at-0 fault in one of the neurons in Path(r). Suppose the faulty neuron is in stage x, then $neuron\,(x, r + 2l \times y_x^r)$ is the faulty one.

Case 3: The network diverges when each memory pattern is applied.

There is a stuck-at-1 faulty neuron which is not in Path(k) where k = 1 to l. The neuron is the one whose output is always one when each memory pattern is applied.

Case 4: The network converges when each memory pattern is applied.

There is either no faulty neuron or a stuck-at-0 neuron which is not in Path(k) where k = 1 to l. In either case, no memory is lost.

In the first three cases, if the fault has been located, the network can be restored by replacing the faulty neuron with a spare fault-free neuron. This fault-recovery method, shown in Figure 6-4, is to add one spare row of neurons to the original network. Suppose that only one neuron in each stage can be faulty, one spare row is sufficient to recover the faults. The recovery process is also shown in Figure 6-4 where the stuck-at-0 neuron is removed and a new minimum path passing through the spare neuron in stage x is assigned.

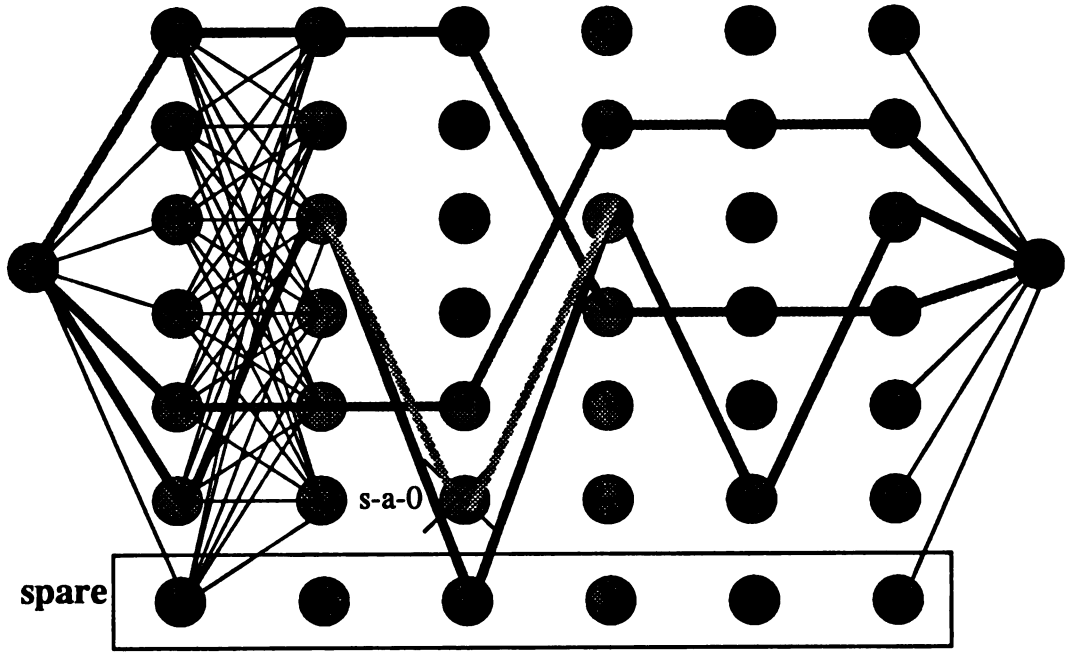


Figure 6-4. Fault recovery process. The thick line represents the new minimum cost path.

6.5 Comparisons

Table 6-4 shows the comparisons between the proposed network and two other networks proposed by Kosko [38] and Wang, *et al* [39]. The network proposed by Wang, *et al.*, which used the multiple training encoding strategy, is an enhanced version of Kosko's network. The data shown in Table IV was obtained from [39]. In both networks, the maximum number of stored patterns is limited. For example, the maximum number of stored patterns is 8 for Kosko's network and 11 for Wang's network when the length of the stored patterns is 100 bits. The stored patterns correspond to local minima in both Kosko's and Wang's networks, while they correspond to global minima in our network.

Both other networks will fail to work if any neuron is damaged. The proposed network can continue to work in general if the simple fault recovery mechanism discussed

in Section 6.4 is adopted. One other major shortcoming for both Kosko and Wang's network is that if additional memory patterns are to be stored, the network connection matrix must be recalculated. One the other hand, the additional patterns can be easily added incrementally in this proposed network without any modification to previous connections provided there are enough spare neurons.

The disadvantage of this network formulation is that the number of neurons required is proportional to the number and length of stored patterns. However, an architecture design using VLSI building blocks for networks of this type has been discussed in Chapter 5. Moreover, only two types of connection weights, 0 or D are required. Thus, it can be envision that this type of structure can ultimately be implemented using cell-based computer-aided design much like the current compilation techniques used to configure memory circuits.

6.6 Summary

In this chapter, an incremental autoassociative memory network based on dynamic programming neural networks was proposed. The formulation of this network is very simple and straightforward in the sense that memory patterns can be easily added to an existing associative memory without any modification to the previous connections. It has been shown that each stored memory pattern corresponds to a global minimum state. Moreover, the inherent ability for fault tolerance makes the proposed network more robust. Simulation has shown that every desired pattern can be successfully stored and recalled. Furthermore, the extent of the basins of attraction of the global minimum state can be adjusted by controlling parameters a and b. Even though there exist some undesired local minimum states, it has been shown that the local minima are all on the boundary of the domain of interest. Two examples for code recovery and pattern recognition have been presented to demonstrate the network's storage ability and flexibility.

Network	Kosko [38] (bipolar mode)	Wang [39] (bipolar mode)	Proposed
N	8	11	not limited
# of neurons required	200	200	N*200
global or local minima	local	local	global
add extra memory	change all connections	change all connections	add extra connections
one neuron damaged	the network will fail	the network will fail	the network can be easily restored
attractivity of memory	unknown	unknown	excellent

Table 6-4. Comparisons of Kosko, Wang, and the proposed network for memory patterns with a length of 100 bits.

Chapter 7

7.0 Application: Dynamic Time Warping in Speech Recognition

This Chapter presents a neural network application for dynamic time warping in speech recognition. The network, based on a feedback-type network for dynamic programming, is designed such that the minimum states of the network's energy function correspond to the near-best correlation between a test and a reference pattern. Simulations for classifying speaker-dependent isolated words, consisting of 0 to 9 and A to Z, show that the recognition rate of the method is as good as that of conventional methods with the same vocabulary set. To achieve real-time applications, a speech recognition system based on this method, which can recognize an utterance in 200 ms with a set of 1,000 references, is also proposed.

7.1 Introduction

To classify human utterances in speech recognition systems, the utterances are generally preprocessed into a set of spectral feature vectors or an acoustic model, and then compared to some pre-stored reference patterns either in a deterministic (template matching approach) or stochastic process (Hidden Markov model) [88,89]. In the template

matching approach, a nonlinear time warping is required to compensate for local compression and/or expansion of the time scales in order to find the best difference measure between the test and reference pattern since their time scales are generally not aligned. Dynamic time warping methods, based on traditional dynamic programming, have been developed to perform time alignment and evaluate the difference between patterns resulting in improved accuracy of speech recognition systems [90-92]. Computational problems, however, exist when the difference measure is computed sequentially. The computation time depends on the length of the feature patterns and the dimension of the search space for evaluating the best difference measures.

Neural networks have been shown to be effective in speech recognition due to the high computation rate attained by the use of many simple processing elements (neurons) operating in parallel and the ability to learn by altering network parameters and connection weights[93,94]. In this chapter, the dynamic programming neural network method is applied to speech recognition of speaker-dependent isolated words. The neural network method for dynamic time warping is introduced in Section 7.2. Simulations and results of recognition of speaker-dependent isolated words, in this case the digits 0 to 9 and English letters A to Z, are shown in Section 7.3. To perform the recognition task in real time, a speech recognition system based on the proposed network is proposed in Section 7.4.

7.2 Dynamic Time Warping

Dynamic programming based on the Principle of Optimality has been an important tool in speech recognition. In the recognition task, the distance between the pattern to be recognized and each of the reference patterns has to be evaluated. Each speech pattern is represented by a set of feature vectors regularly spaced along a time axis. These vectors can be the outputs of a set of filters, LPC coefficients, or cepstral coefficients as will be used in the simulations described in this chapter. Since the time scales of a test and a reference speech pattern are generally not perfectly aligned, alignment becomes necessary in order to

obtain the difference measure. In most cases, a nonlinear time warping technique is required to obtain the optimal alignment since the differences between patterns in time, frequency, and amplitude are nonlinear. Therefore, a class of algorithms known as dynamic time warping (DTW) have been developed to perform the optimal alignment and evaluate the difference between two patterns [90-92].

Suppose that the kth reference pattern, R_k, is a sequence of vectors $R_{k,m}$, where m = 1, 2,..., $M(k)$, and the test pattern T is a sequence of vectors T_n where n = 1, 2,..., N. An example of time warping function $w(n)$ is shown in Figure 7-1 with the boundary conditions [95]

$$w(1) = 1, \quad and \quad w(N) = M(k), \tag{7-1.1}$$

and continuity conditions

$$w(n+1) - w(n) = \begin{cases} 0, 1, 2 \ for \ w(n) \neq w(n-1) \\ 1, 2 \ for \ w(n) = w(n-1) \end{cases} \tag{7-1.2}$$

Equations (7-1.1) and (7-1.2) determine the dimension of the search space and each time warping function produces a specific path in the search space.

Throughout this paper, it is assumed that the beginning and the ending points of the test and reference patterns have been accurately located. The distance between vector T_n and $R_{k,w(n)}$ is defined as

$$d(n, w(n); k) = \frac{\sum_{l=1}^{L} (T_n[l] - R_{k,w(n)}[l])^2}{1024 \times L} \tag{7-2}$$

where L is the dimension of feature vectors, and $T_n[l]$ and $R_{k,w(n)}[l]$ are the lth component of vector T_n and $R_{k,w(n)}$, respectively. The value of every component of the feature vectors is in the range of [0, 1024], and so is every $d(n, w(n), k)$ for all n and k. The total distance

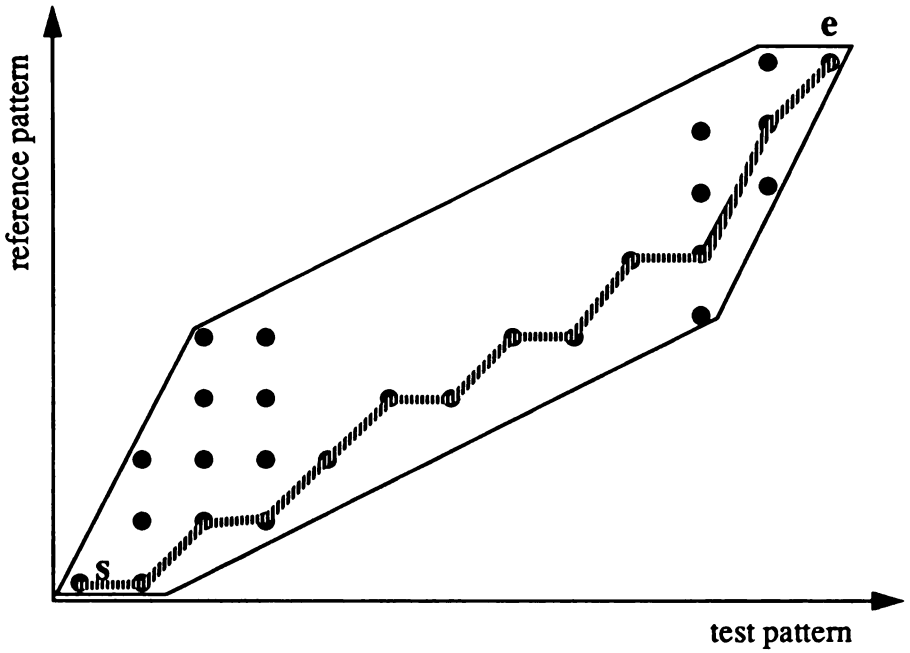


Figure 7-1 An example of the time warping function and search space of the optimal path. The path corresponding to this warping function is shown by the hatched line.

for the path resulting from the time warping function, that starts at grid point (1, 1) and ends at grid point (N, $M(k)$), can be defined as a sum of local distance $d(n, w(n); k)$ for each (n, $w(n)$) grid point of the path. Then the problem of dynamic time warping is to find an optimal path which minimizes the sum of $d(n, w(n); k)$ among all possible time warping functions. The distance of the optimal path is expressed as

$$D(k) = \min_{w(\cdot)} \sum_{n=1}^{N} d(n, w(n); k) = \min_{w(\cdot)} \sum_{n=1}^{N} \frac{\sum_{l=1}^{L} (T_n[l] - R_{k, w(n)}[l])^2}{1024 \times L} . \tag{7-3}$$

Define the partial distance of $D(k)$ from time instant $j = 1$ to n as

$$D(n, w(n); k) = \min_{w(\cdot)} \sum_{j=1}^{n} d(j, w(j); k) . \tag{7-4}$$

The distance $D(k)$ in the example can be efficiently computed, using the algorithm of dynamic programming, by the following recursive equation

$$D(n+1, w(n+1); k) = d(n+1, w(n+1); k) + \min\{D(n, w(n+1); k) \times g(n)), D(n, w(n+1) - 1; k), D(n, w(n+1) - 2; k)\} \tag{7-5.1}$$

where

$$g(n) = \begin{cases} \infty & for \ w(n) = w(n-1) \\ 1 & for \ w(n) \neq w(n-1) \end{cases} \tag{7-5.2}$$

such that the continuity conditions of equation (7-1.2) are satisfied [90]. In equation (7-5.1), it is assumed that $d(n, m; k)$ outside the search space is infinitely large.

Notice that the computation of the conventional dynamic time warping algorithm is executed sequentially, therefore the computation time depends on the length of feature

patterns and the dimension of the search space. On the other hand, the following method based on a dynamic programming neural network, performs distance calculations fully in parallel. Thus the computation time is significantly reduced and is much less dependent on the length of patterns and the dimension of the search space.

7.2.1 A Neural Network for Dynamic Time Warping

Consider Figure 7-1 again. Assume each warping function $w(n)$ has one and only one value for each time instant. A valid and optimal path associated with the test pattern T and the kth reference pattern R_k is the one which starts at grid point (1, 1), visits one and only one grid point in each time instant, reaches grid point (N, $M(k)$), and has a minimum total distance among all possible paths.

Each grid point can be conveniently considered as an individual processing element. To develop an appropriate energy function for the network, v_{ni} is taken as the output of a processing element in the position (n, i); the weight for the connection $c_{ni,(n+1)j}$ between the grid point (n, i) and ($n+1$, j) is defined as $w_{ni,(n+1)j} = d(n+1, j; k)$ for $j - i = 0$, 1, and 2, or infinity. The following formal constraints are thus defined.

Following the same procedure as in Chapter 3, the neural network for dynamic time warping can be described by the following simplified differential equation

$$C_{xi}\left(\frac{du_{xi}}{dt}\right) = -\alpha\sum_{j\neq i} v_{xj} - \alpha\sum_{y}\sum_{j} v_{yj} + \alpha n - \frac{\beta}{2}\sum_{j}\left(d_{xi,(x+1)j}v_{(x+1)j} + d_{(x-1)j,xi}v_{(x-1)j}\right) \qquad (7\text{-}9.1)$$

where

$$\frac{1}{R_{ni}} = \frac{1}{\rho_{ni}} + \sum_{rj} w_{ni,rj}. \qquad (7\text{-}9.2)$$

7.3 Network Simulation

Although it is not guaranteed that the networks can provide the globally optimal paths, simulation results in Chapter 4 have shown that the generated paths are very close to the optimal paths. The distances for the selected paths can then be used as difference measures to classify speech patterns.

7.3.1 Vocabulary Set

The vocabulary set consists of 36 isolated words which are the letters A -Z and the digits 0 - 9. Each word was spoken 15 times by a male speaker with the first 5 utterances for reference patterns and the other 10 utterances for test patterns. The number of segments of utterances is in the range of 30 to 50 with each segment corresponding to 10 to 20 ms of acoustic signal. Each segment is represented by a 6-dimensional vector of cepstral coefficients.

7.3.2 Initial Conditions

The selection of initial conditions for the networks is very important since there is more than one equilibrium point in each associated energy function. Experiments have shown that the optimal path usually occurs near the diagonal connecting the grid point (1, 1) and (N, $M(k)$) [95]. Moreover, the grid points with larger local distance should be penalized to prevent them from being selected as valid paths. Suppose the diagonal can be expressed as $y = f(x; k)$. The initial condition is defined as

$$v_{nm} = 0.05 \times |m - f(n;k)| \times \delta_1 (3 - |m - f(n;k)|) + \frac{300 - d(n, m;k)}{3000} \times \delta_1 (300 - d(n, m;k)) + 0.05 \qquad (7\text{-}10)$$

where $\delta_1 (c) = 1$ if $c \geq 0$, and otherwise 0. Since each value of $d(n, m; k)$ belongs to [0, 1024] in the simulation, v_{nm} is in the range of [0.05, 0.3] for all n, and m.

7.3.3 Results

In the simulation, the parameters α and β are set to 25 and 0.01, respectively. To simplify the calculations, each $d(n, m; k)$ is rounded to the nearest integer. Three methods are simulated. The first uses the conventional time warping method described in Section 7.2 with one reference pattern. The other two, based on the neural network method for dynamic programming described in Section 7.2.1, use one and five reference patterns, respectively. The method with five reference patterns follows a similar classification procedure as that of the 5-nearest-neighbor method [46]. The threshold to classify a test pattern into a category is 2. Therefore, this method classifies a test pattern to the category which has more than three best scores. In the case of a tie, where two categories have 2 best scores among the best five, a test pattern will be classified into the category which has the lowest sum of scores. Ten test patterns are used for each word to test the performance of the networks. Therefore there are 360 test patterns in the simulation.

The recognition rates for these three methods are 87.22%, 86.66%, and 92.22%. Note that the first two results are very close. Table 7-1 compares some distances found by the conventional method which obtains the optimal path distances and the neural network method for dynamic time warping with only one reference for a test utterance of A. In this example, it happens that both methods correctly classify the test pattern. However, it is not always true for every test pattern. In some cases, misclassifications occur in either method or both. A similar recognition rate for the same vocabulary set and using LPC coefficients obtained by Itakura is 88.60% for 720 utterances [90]. This is not surprising since they all are based on dynamic time warping. Notice that the one with 5 reference patterns has much better recognition rate than that with only one reference pattern. This fact affirms that the k-nearest-neighbor method is superior to the nearest-neighbor method with appropriate value of k. However, the penalty is the increase in classification time.

Reference	A	B	C	D	E	F	G	H	I
Conventional	432	463	5534	549	2247	1711	6398	2107	1940
Neural net.	704	750	6187	739	2565	2490	9020	2158	2590

Table 7-1 Comparisons of distances obtained by the conventional and neural network method for DTW for one of the test pattern A.

All digits are correctly recognized in the simulation. Most of the misclassifications occur in utterances where the vowel part is identical and the difference of the consonant part is relatively small. For example, B is often misclassified as D or V. Major misclassifications for the neural network methods are listed in Table 7-2. It is worthy to notice that the E test utterances are totally misclassified if only one reference pattern is used. However, only 2 errors occur when 5 reference patterns are used.

7.4 A Speech Recognition System Based on the Proposed Network

Dynamic time warping is often implemented either on general digital signal processors or some specially designed processors [96-99]. In this section, we propose a neural network architecture for dynamic time warping with three salient advantages. The first is the high computation rate due to many simple processing elements operating in parallel, a typical feature of artificial neural networks.The second one is the attribute of hardware fault tolerance due to the nature of collective computation in neural networks. The network can still provide good solutions even in situations where some of the processing elements fail to work. This is primarily due to the fact there is more than one minimum state in the associated energy function. The third advantage comes from the inclusion of simple processing elements and a regular structure of this architecture making implementation of this architecture more feasible with current VLSI technology.

Test Pattern	Misclassified as	Number of Errors	
		1 ref.	5 ref.
B	D or V	4	2
D	B or V	5	5
E	B	10	2
I	Y	2	1
N	M	2	2
T	D or P	5	5
others		19	11
	Total errors	48	28

Table 7-2 Major misclassifications of the simulation for the neural network methods. The second column lists the misclassified patterns for some of the test patterns in the first column. The third column lists the number of misclassifications. For example, B is misclassified 4 and 2 times in 10 attempts as D or V in the methods with one and five reference patterns, respectively.

Figure 7-2 shows the block diagram of a speech recognition system including the artificial neural network for dynamic time warping. This system consists of four major parts: a signal preprocessor, a pipelined distance calculator, a control unit, and the artificial neural network itself. An utterance signal is preprocessed by the signal processor into a sequence of feature vectors of cepstral coefficients. The local distances, $d(n, m; k)$, for a test pattern and a reference pattern are calculated by the pipelined distance calculator in order to speed up the distance calculations. The calculated distances are then sent by the control unit to the neural network to set the connection weights of the network. The outputs of the network are fed back to the control unit to determine the difference measure between the test and reference pattern.

The pipelined distance calculator, shown in Figure 7-3, has 6 stages corresponding to the case where the dimension of feature vectors is 8. Four stages are for addition (subtraction), one for multiplication, and the last stage is for division. This last stage can be

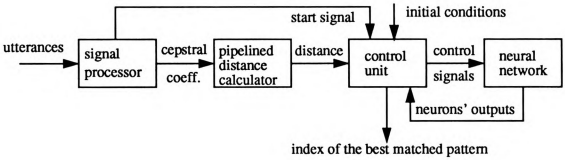


Figure 7-2. The block diagram of the speech recognition system which includes the neural network for dynamic time warping.

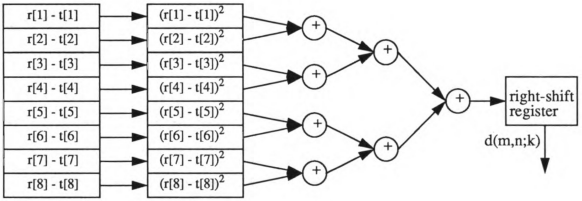


Figure 7-3. The pipelined local distance calculator.

simply implemented by a shift-right register since the denominator is 2^{13}. The control unit sets the initial conditions of the processing elements, determines the weights of the connections, and selects the best matched reference pattern.

To give an approximate estimation of the processing speed of the system, we assume that the cepstral coefficients of the test pattern have been obtained by the signal preprocessor, and the processing time spent in the control unit can be neglected. Let the maximum delay for a stage of the pipelined distance calculator be T_{delay}, the number of grid points in the search space be N_g, and the convergence time for the neural network be T_c. Then the total processing time to classify a test pattern is approximately equal to

$$K \times ((N_g + 5) \times T_{delay} + T_c) \tag{7-11}$$

where K is the number of reference patterns. The stage with the maximum delay of the distance calculator is the multiplication stage, and the convergence time of the neural network is dependent on the RC constants of the network.

A 16-by-16 multiplier, which can be constructed by 4-by-4 binary multipliers (SN74S274) and look-ahead adders (SN74S182), has a typical multiplication time less than 150 *ns* [100]. Assume the length of the feature vectors is 50, then the number of grid points in the search space is approximately equal to 500 [90]. From the experiments we have conducted, if RC is set to 1/2000, a network requires approximately 100 *us* to converge. If a recognition system has 1000 reference patterns, then the processing time to classify a test pattern is approximately equal to 200 *ms*, well within the limits for real-time applications.

7.5 Summary

A neural network method for dynamic time warping in speech recognition has been proposed. Simulations for classifying speaker-dependent isolated words have shown that the recognition rate of the method is as good as that of conventional methods with the same

vocabulary set. The recognition rate can be improved if more reference patterns are used. An architecture based on this method has also been proposed. There are three advantages to this proposed architecture. The first is the high computation rate due to many simple processing elements operating in parallel. The second is the attribute of fault tolerance to hardware faults due to the nature of collective computation of the neural network. And the third advantage is that this architecture is simple and regular, making implementation of this architecture feasible using current VLSI technology.

Chapter 8

8.0 Conclusion

Dynamic programming is a very useful technique in many engineering applications, such as optimal control, optimization, and speech recognition. Due to the curse of dimensionality and the sequential computation nature, conventional algorithms are not effective in many applications requiring fast, perhaps real-time, solutions. This dissertation presents a fundamentally new approach to dynamic programming, including network formulation, analysis, simulation, implementation, and applications. The described artificial neural network method is attractive due to its robustness and radically improved speed over conventional techniques especially where real-time near-optimal solutions are required.

8.1 Summary

This dissertation has presented an intensive study of applying neural networks to dynamic programming problems. The network formulation was defined by an energy function in which the optimal solution corresponds to the lowest energy state of the network. Following the development of the energy function, the network equations were determined. The network formulation has been shown to be a gradient system which converges to one of the stable equilibria if the initial conditions are sufficiently close.

The formulated network was analytically examined from the view point of large-

scale nonlinear systems. Properties related to the functionality of the network were proved. These include the existence and uniqueness of the solution, the stability and convergence of the network, the relationship between the equilibria and minima of the energy function, and the finite number of equilibria of the network. To gain a clearer insight into the nonlinear system, each component of the associated energy function was discussed in the extreme cases. With $a \rightarrow \infty$, the minimum states are the valid corner states representing valid solutions. With $a = 0$, the zero state is the unique minimum state. Then, the possible locations of the minimum states of the associated energy function for different parameter values a and b were derived.

Simulations have shown that the quality of the solutions improves as the value of parameter b increases up to a point. However, invalid results may occur if b is greater than a certain threshold. The network may become unstable or the minimum states may move away from their original locations if the connection errors exceeds a certain value. In general, the smaller the corresponding total path of a minimum state is, the more likely the minimum state can be preserved in its associated valid region. The basins of attraction of the network's equilibria were also investigated. It has been shown that the number of attracted initial conditions, which can be used as an index of the basins of attraction, increases as b is increased. Moreover, results indicate that the formulated network can provide a near-optimal solution for different problems with sizes as large as 64 stages with 64 states in each stage.

An architecture based on a building block paradigm, in which the network is constructed from neuron array and weight assignment chips, was described. Because of its simple and regular structure, the architecture is a feasible implementation with current VLSI technology.

Two significant applications based on dynamic programming neural networks were described in this dissertation as case studies. First, an incremental autoassociative memory network, in which memory patterns can be easily added to an existing associative memory without any modification to the previous connections, was proposed. It has been proven that each stored memory pattern corresponds to a global minimum state. Simulation has shown that every desired pattern can be successfully stored and recalled. Furthermore, the extent of the basins of attraction

of the global minimum state can be adjusted by controlling parameters a and b. Comparison with two other networks from the literature shows that the incremental autoassociative memory network is more flexible and robust.

The second application is dynamic time warping in speech recognition. The network was designed such that the minimum states of the energy function correspond to the near-best correlation between a test and a reference pattern. Simulations for classifying speaker-dependent isolated words, consisting of 0 to 9 and A to Z, have shown that the recognition rate of the method is as good as that of conventional methods with the same vocabulary set. Moreover, a speech recognition system based on this neural network method, which can recognize an utterance in 200 ms with a set of 1,000 references, was proposed.

8.2 Contributions

This dissertation has the following salient contributions:

(1) This dissertation has presented a fundamentally new and different approach to dynamic programming. This artificial neural network method is attractive due to its robustness and radically improved speed over conventional techniques.

(2) This dissertation has provided a new analysis method for neural networks from the energy function point of view. These analytical results lay a mathematical foundation for the future research.

(3) This dissertation has successfully applied the dynamic programming neural networks to different applications such as optimal control, autoassociative memory, speech recognition, and flight planning.

APPENDICES

APPENDIX A

The 50x60 cost values:

0.000000 0.084274 0.078459 0.068828 0.044160 0.014342 0.004604 0.004604
0.007609 0.014133 0.034845 0.102514 0.112155 0.108527 0.086704 0.060406
0.042231 0.005931 0.017125 0.018539 0.038956 0.111514 0.132861 0.141905
0.124018 0.097794 0.080150 0.041049 0.044340 0.020466 0.039012 0.111668
0.133011 0.141958 0.124021 0.102154 0.083596 0.042705 0.046155 0.020466
0.039012 0.111668 0.133011 0.142258 0.124221 0.102438 0.083758 0.042992
0.046453 0.020653 0.039210 0.111930 0.134162 0.142304 0.124221 0.102438
0.083758 0.043160 0.046601 0.020745

0.000000 0.078297 0.085005 0.058051 0.038992 0.009139 0.009139 0.009139
0.009139 0.013671 0.019753 0.090301 0.091803 0.099115 0.084584 0.051351
0.011106 0.005291 0.011485 0.018510 0.019753 0.098548 0.095402 0.133081
0.121798 0.087752 0.048353 0.040422 0.041041 0.020563 0.020054 0.098789
0.095651 0.133121 0.140290 0.109107 0.060448 0.043609 0.043095 0.020563
0.020054 0.098789 0.095651 0.133121 0.140471 0.110876 0.061522 0.043877
0.043347 0.020818 0.020236 0.099391 0.095787 0.133121 0.140471 0.110876
0.061522 0.043910 0.043447 0.022108

0.000000 0.065918 0.047892 0.032700 0.018501 0.018501 0.018501 0.018501
0.018501 0.018501 0.031112 0.078490 0.052349 0.074612 0.079573 0.012063
0.029771 0.003365 0.001970 0.000951 0.031239 0.083819 0.071379 0.104705
0.114286 0.047060 0.036348 0.040704 0.006716 0.007825 0.031523 0.083872
0.071427 0.113629 0.138166 0.113242 0.078295 0.048324 0.008667 0.008854
0.031523 0.083872 0.071427 0.113629 0.138302 0.113567 0.078495 0.048604
0.008895 0.009066 0.032291 0.084066 0.071427 0.113629 0.138302 0.113567
0.078495 0.048604 0.009186 0.009891

0.095740 0.024221 0.025691 0.013847 0.008064 0.008064 0.008064 0.008064
0.008064 0.008064 0.021106 0.035518 0.029386 0.018773 0.045479 0.022924
0.016031 0.009240 0.004304 0.004002 0.021124 0.039101 0.040989 0.043687
0.077465 0.053312 0.047119 0.035041 0.007469 0.004574 0.021127 0.039101
0.040989 0.066690 0.122496 0.094759 0.084801 0.073846 0.011204 0.012461
0.021852 0.039101 0.040997 0.066740 0.122733 0.094988 0.084901 0.074097
0.013410 0.012547 0.021880 0.039101 0.040997 0.066740 0.122733 0.094988
0.084901 0.074097 0.013410 0.012817

0.084196 0.008955 0.001701 0.001701 0.001701 0.001701 0.001701 0.001701
0.001701 0.001701 0.010239 0.013617 0.008643 0.006131 0.018624 0.013942
0.014948 0.008470 0.004492 0.004175 0.010309 0.014043 0.014882 0.020080
0.022032 0.035962 0.035928 0.011133 0.005481 0.004341 0.010374 0.018089
0.082452 0.061164 0.079141 0.131332 0.101828 0.045423 0.008961 0.005942
0.010374 0.018089 0.082465 0.061231 0.079410 0.131669 0.102035 0.045789
0.009334 0.006072 0.010374 0.018089 0.082465 0.061231 0.079410 0.131669
0.102035 0.045789 0.009334 0.006072

0.079307 0.079307 0.079307 0.079307 0.079307 0.079307 0.079307 0.079307
0.079307 0.079307 0.079307 0.001504 0.001664 0.003166 0.003734 0.003317
0.003722 0.001779 0.001067 0.000816 0.000985 0.001587 0.001764 0.007138
0.010230 0.010728 0.004610 0.002269 0.001519 0.000983 0.005858 0.009982
0.018336 0.063641 0.078575 0.027667 0.014481 0.007524 0.003569 0.000983
0.005858 0.009985 0.018355 0.064170 0.078843 0.028146 0.014745 0.007819
0.003909 0.001113 0.005858 0.009985 0.018355 0.064170 0.078843 0.028146
0.014745 0.007819 0.003909 0.001113

0.074540 0.074540 0.074540 0.074540 0.074540 0.074540 0.074540 0.074540
0.074540 0.074540 0.074540 0.074540 0.074540 0.074540 0.002125 0.002196
0.001317 0.000496 0.000330 0.000519 0.000634 0.002134 0.001621 0.001202
0.004060 0.004477 0.001511 0.001702 0.000562 0.000785 0.002441 0.006723
0.011156 0.046189 0.079500 0.017260 0.001511 0.001702 0.000562 0.000785
0.002441 0.006723 0.011194 0.046317 0.079867 0.017833 0.001649 0.001800
0.000680 0.000818 0.002441 0.006727 0.011194 0.046317 0.079867 0.017833
0.001649 0.001800 0.000680 0.000818

0.089847 0.089847 0.089847 0.089847 0.089847 0.089847 0.089847 0.089847
0.089847 0.089847 0.089847 0.001582 0.001582 0.001582 0.001436 0.001436
0.001436 0.001436 0.001436 0.000840 0.000581 0.000957 0.001808 0.001808 0.001808
0.002554 0.002554 0.002554 0.003443 0.001050 0.000845 0.001096 0.002891
0.005736 0.008696 0.007455 0.004714 0.004714 0.003443 0.002202 0.000845
0.001096 0.002908 0.005802 0.009602 0.007760 0.004991 0.003652 0.003499
0.002309 0.000879 0.001096 0.002908 0.005802 0.009602 0.007760 0.004991
0.003652 0.003499 0.002309 0.000879

0.089367 0.089367 0.089367 0.089367 0.089367 0.089367 0.089367 0.089367
0.089367 0.089367 0.089367 0.001567 0.000687 0.000687 0.000687 0.000687
0.000251 0.000413 0.000326 0.000494 0.001775 0.003315 0.001771 0.001771
0.001771 0.001119 0.000396 0.000718 0.000655 0.000873 0.001995 0.004231
0.003756 0.006838 0.004328 0.001180 0.000396 0.000718 0.000655 0.000873
0.001996 0.004259 0.003909 0.007110 0.004503 0.001435 0.001278 0.000863
0.001953 0.000922 0.001996 0.004259 0.003909 0.007110 0.004503 0.001435
0.001278 0.000863 0.001953 0.000922

0.096186 0.104923 0.142991 0.139899 0.123460 0.093852 0.044177 0.051987
0.027380 0.021356 0.039847 0.106748 0.143206 0.139971 0.123507 0.093921
0.046200 0.052235 0.027479 0.024214 0.039989 0.106785 0.143207 0.140299
0.123623 0.094110 0.046389 0.053425 0.027684 0.024637 0.040350 0.106988
0.143339 0.140966 0.125173 0.094286 0.046487 0.053489 0.027743 0.024709
0.040439 0.108220 0.143653 0.143040 0.126656 0.094639 0.046665 0.053637
0.028049 0.025566 0.040445 0.108220 0.143653 0.143040 0.126656 0.094639
0.046665 0.053637 0.028049 0.025566

0.074625 0.113498 0.134280 0.142469 0.124337 0.102448 0.083758 0.043160
0.046601 0.020745 0.039501 0.113721 0.134400 0.143201 0.133389 0.106048
0.087341 0.046488 0.052456 0.024666 0.040583 0.114253 0.134449 0.143201
0.133819 0.106195 0.087569 0.046672 0.052620 0.025012 0.040961 0.114555
0.134709 0.143830 0.133936 0.107998 0.087802 0.046803 0.052738 0.025196
0.041180 0.116807 0.136736 0.144040 0.134329 0.108437 0.088521 0.047662
0.053017 0.025350 0.041180 0.116807 0.136736 0.144040 0.134329 0.108437
0.088521 0.047913 0.053023 0.025377

0.075134 0.101420 0.095907 0.134315 0.141322 0.110974 0.061522 0.043910
0.043447 0.022108 0.021378 0.101420 0.095941 0.135529 0.145395 0.126147
0.086452 0.048697 0.051433 0.025200 0.023065 0.102345 0.096234 0.135529
0.145395 0.126200 0.088035 0.048942 0.051521 0.025377 0.023304 0.102509
0.097863 0.135713 0.146094 0.126329 0.088215 0.049067 0.051618 0.025444
0.023454 0.102656 0.097959 0.136141 0.147216 0.126750 0.089733 0.049199
0.051731 0.025444 0.023454 0.102656 0.097959 0.136141 0.147216 0.126819
0.091398 0.049593 0.052151 0.025444

0.088932 0.085376 0.071478 0.114632 0.138420 0.113645 0.078495 0.048604
0.009186 0.009891 0.033774 0.085376 0.071478 0.114812 0.142974 0.121308
0.085924 0.054358 0.027330 0.015630 0.036933 0.085819 0.071777 0.114812
0.142974 0.121308 0.086052 0.054462 0.027375 0.015664 0.036997 0.085899
0.071899 0.116411 0.144107 0.121378 0.086164 0.054533 0.027488 0.015745
0.037034 0.085968 0.072091 0.116626 0.144285 0.121613 0.086410 0.054619
0.027488 0.015745 0.037034 0.085968 0.072124 0.117036 0.145507 0.130535
0.087936 0.055993 0.032256 0.015970

0.091389 0.040357 0.041684 0.068088 0.122892 0.095079 0.084901 0.074097
0.013410 0.012817 0.022078 0.040357 0.041684 0.072424 0.132912 0.116698
0.146721 0.086507 0.018820 0.016978 0.023929 0.041328 0.042153 0.072431
0.132912 0.116698 0.146773 0.086507 0.018820 0.016999 0.023975 0.041385
0.042281 0.072647 0.133180 0.116939 0.146957 0.086507 0.018820 0.016999
0.024044 0.041453 0.042469 0.072806 0.133312 0.117112 0.147021 0.086507
0.018820 0.016999 0.024044 0.041524 0.042621 0.073858 0.135326 0.120654
0.151169 0.105772 0.035364 0.018110

0.065059 0.019289 0.082968 0.062783 0.081108 0.131742 0.102035 0.045789
0.009334 0.006072 0.011181 0.019289 0.083551 0.066265 0.092754 0.210739
0.154901 0.090700 0.059835 0.015111 0.014591 0.028154 0.084261 0.070244
0.092754 0.210739 0.154901 0.090700 0.059835 0.015125 0.014623 0.028200
0.084337 0.070346 0.092850 0.210884 0.155013 0.090700 0.059835 0.015125
0.014623 0.028200 0.084452 0.070416 0.092884 0.211148 0.155013 0.090700
0.059835 0.015125 0.014623 0.028270 0.085198 0.073440 0.097452 0.217360
0.188825 0.124065 0.086112 0.030682

0.059192 0.011365 0.018425 0.065374 0.079084 0.029137 0.015159 0.007819
0.003909 0.001113 0.005908 0.011440 0.021838 0.071727 0.095355 0.064497
0.054709 0.047702 0.041985 0.038599 0.008802 0.022501 0.026781 0.072783
0.095355 0.064497 0.054775 0.047702 0.041985 0.038602 0.008809 0.022531
0.026812 0.072813 0.095371 0.064539 0.054821 0.047702 0.041985 0.038602
0.008809 0.022531 0.026838 0.073134 0.096962 0.066890 0.055162 0.047702
0.041985 0.038602 0.008809 0.022761 0.032459 0.077636 0.103061 0.103206
0.090400 0.082687 0.075780 0.059898

0.087031 0.008631 0.012501 0.047269 0.080993 0.018009 0.001791 0.001846
0.000680 0.000818 0.002490 0.008631 0.012501 0.054877 0.093342 0.043162
0.038811 0.041978 0.014666 0.029396 0.016179 0.010026 0.016472 0.054877
0.093342 0.043162 0.038940 0.042209 0.014829 0.029525 0.016306 0.010089
0.016488 0.054887 0.093353 0.043179 0.038940 0.042209 0.014829 0.029525
0.016306 0.012688 0.020454 0.062141 0.102047 0.052207 0.043750 0.042409
0.014829 0.029525 0.016306 0.013014 0.028045 0.083752 0.134096 0.087636
0.080198 0.079019 0.049327 0.057656

0.073765 0.004105 0.007291 0.011094 0.008751 0.005700 0.004785 0.003676
0.003106 0.000933 0.001675 0.004157 0.008082 0.017533 0.018797 0.022157
0.028976 0.010188 0.006952 0.002985 0.003010 0.009079 0.011137 0.017533
0.018797 0.022187 0.030298 0.012125 0.007749 0.004068 0.003302 0.009203
0.011209 0.017545 0.018799 0.022187 0.030301 0.012129 0.007749 0.004068
0.003302 0.015053 0.023259 0.036834 0.042815 0.041243 0.047361 0.013460
0.008224 0.004097 0.003302 0.015053 0.024276 0.056289 0.076209 0.077581
0.085022 0.050051 0.042550 0.008519

0.057531 0.005027 0.004842 0.007411 0.006030 0.001863 0.001417 0.001142
0.002213 0.002282 0.002198 0.006177 0.006476 0.011748 0.011158 0.003242
0.003590 0.002511 0.002809 0.003359 0.011633 0.006914 0.007062 0.011748
0.011187 0.003356 0.003788 0.002700 0.002858 0.003505 0.011910 0.007575
0.007251 0.011748 0.011187 0.003356 0.003788 0.002703 0.002858 0.003505
0.014819 0.020207 0.031794 0.043173 0.044164 0.037296 0.029755 0.018699
0.007715 0.003534 0.014819 0.020259 0.035568 0.045513 0.069374 0.071268
0.064554 0.053220 0.036960 0.007050

0.040026 0.108685 0.143757 0.143217 0.126837 0.095779 0.047268 0.054559
0.029453 0.026731 0.040563 0.109361 0.143905 0.143613 0.127936 0.101525
0.048228 0.060819 0.030344 0.028010 0.041548 0.109680 0.144937 0.143635
0.128047 0.101727 0.049095 0.061472 0.030491 0.028206 0.041716 0.109757
0.145161 0.143642 0.128047 0.101727 0.049095 0.061472 0.030491 0.028415
0.046720 0.128485 0.176477 0.178744 0.165046 0.138163 0.084097 0.065296
0.039891 0.028678 0.046720 0.128505 0.178314 0.187154 0.183469 0.160480
0.111180 0.090787 0.059823 0.038243

0.047013 0.117183 0.136810 0.144576 0.135351 0.110064 0.089556 0.047992
0.054027 0.026606 0.042427 0.117986 0.137684 0.145172 0.135562 0.110433
0.095218 0.052736 0.055827 0.026653 0.042459 0.118032 0.137738 0.145282
0.135686 0.110611 0.095329 0.052873 0.056015 0.026865 0.042587 0.118075
0.137761 0.145292 0.135686 0.110611 0.095329 0.052873 0.056015 0.027079
0.050892 0.140764 0.170776 0.182250 0.172903 0.148729 0.130135 0.079020
0.065624 0.028882 0.050892 0.140764 0.171657 0.184366 0.180301 0.160796
0.144143 0.094425 0.073872 0.034107

0.041253 0.102995 0.098010 0.136720 0.148225 0.127043 0.093088 0.049814
0.053680 0.027087 0.025608 0.104472 0.099459 0.138320 0.150196 0.128968
0.094932 0.053199 0.053984 0.027330 0.025718 0.104563 0.099624 0.138565
0.150356 0.129092 0.096500 0.053474 0.054142 0.027440 0.025820 0.104611
0.099644 0.138604 0.150356 0.129092 0.096500 0.053474 0.054142 0.027440
0.032096 0.124894 0.133494 0.174308 0.188385 0.167040 0.130805 0.076223
0.055975 0.027914 0.032126 0.124894 0.133494 0.174311 0.189647 0.170413
0.136814 0.078218 0.058205 0.028983

0.028983 0.086133 0.072167 0.117093 0.146094 0.131324 0.088969 0.056150
0.033415 0.016084 0.037198 0.086859 0.073255 0.118597 0.146539 0.131541
0.089098 0.056339 0.033698 0.016274 0.037337 0.086953 0.073255 0.118709
0.146666 0.131644 0.089347 0.058012 0.033825 0.016322 0.037402 0.086998
0.073283 0.118713 0.146666 0.131644 0.089347 0.058012 0.033825 0.016322
0.041184 0.089605 0.099125 0.151874 0.153545 0.170811 0.118776 0.081311
0.034687 0.016452 0.041184 0.089605 0.099125 0.151874 0.153545 0.171295
0.119203 0.081311 0.034687 0.016452

0.008839 0.041566 0.042640 0.073895 0.135505 0.120885 0.151201 0.106167
0.036072 0.018172 0.027109 0.041948 0.042771 0.074637 0.135642 0.120990
0.151311 0.106205 0.036177 0.019206 0.027241 0.042024 0.042771 0.074637
0.135704 0.121129 0.151438 0.106991 0.036257 0.019268 0.027283 0.042119
0.042848 0.074673 0.135704 0.121129 0.151438 0.106991 0.036257 0.019272
0.029080 0.050554 0.060769 0.096649 0.162990 0.147307 0.155325 0.108597
0.036442 0.019273 0.029080 0.050554 0.060769 0.096649 0.162990 0.147307
0.155325 0.108597 0.036442 0.019273

0.019273 0.028270 0.085204 0.073456 0.097540 0.217480 0.188841 0.124287
0.086159 0.030723 0.019661 0.028652 0.085772 0.073541 0.097562 0.217563
0.188844 0.124429 0.086353 0.030978 0.019760 0.028826 0.085772 0.073541
0.097701 0.217821 0.188896 0.124510 0.086422 0.031052 0.019825 0.029074
0.086038 0.073662 0.097701 0.217821 0.188896 0.124521 0.086422 0.031071
0.020523 0.031002 0.093305 0.085795 0.112160 0.233943 0.190266 0.125467
0.086422 0.031071 0.020523 0.031002 0.093305 0.085795 0.112160 0.233943
0.190266 0.125467 0.086422 0.031071

0.031071 0.022761 0.032459 0.077638 0.103061 0.103207 0.090400 0.082687
0.075782 0.059906 0.010543 0.022819 0.032790 0.077677 0.103197 0.103207
0.090532 0.082764 0.075946 0.060039 0.010682 0.022929 0.032790 0.077677
0.103270 0.103315 0.090577 0.082802 0.075977 0.060101 0.010770 0.023272
0.032848 0.077982 0.103575 0.103544 0.090656 0.082914 0.076041 0.060215
0.010815 0.023272 0.034040 0.078579 0.104561 0.103848 0.091061 0.083988
0.076041 0.060215 0.010815 0.023272 0.034040 0.078579 0.104561 0.103848
0.091061 0.083988 0.076041 0.060215

0.060215 0.013014 0.028045 0.083752 0.134096 0.087636 0.080198 0.079019
0.049327 0.057658 0.018233 0.013030 0.028056 0.083837 0.134103 0.087652
0.080247 0.079047 0.049429 0.057733 0.018278 0.013059 0.028056 0.083837
0.134103 0.087681 0.080274 0.079078 0.049469 0.057806 0.018367 0.013111
0.028557 0.084311 0.134757 0.088132 0.080472 0.079324 0.049617 0.057847
0.018588 0.013118 0.028557 0.084757 0.135099 0.088132 0.080472 0.079324
0.049617 0.057847 0.018588 0.013118 0.028557 0.084757 0.135099 0.088132
0.080472 0.079324 0.049617 0.057847

0.000000 0.015053 0.024276 0.056289 0.076209 0.077581 0.085022 0.050051
0.042550 0.008519 0.004797 0.015053 0.024276 0.056298 0.076209 0.077581
0.085030 0.050071 0.042613 0.008554 0.004824 0.015079 0.024276 0.056303
0.076209 0.077609 0.085065 0.050118 0.042736 0.008749 0.004959 0.015168
0.025359 0.057238 0.077264 0.077775 0.086000 0.050877 0.043252 0.008833
0.005397 0.015193 0.025405 0.057238 0.077264 0.077775 0.086000 0.050877
0.043252 0.008833 0.005397 0.015259 0.025405 0.057238 0.077264 0.077775
0.086000 0.050877 0.043252 0.008833

0.000000 0.020259 0.035568 0.045513 0.069374 0.071268 0.064554 0.053220
0.036960 0.007050 0.020442 0.020259 0.035568 0.045515 0.069374 0.071269
0.064554 0.053220 0.037042 0.007060 0.020442 0.020259 0.035568 0.045525
0.069390 0.071293 0.064617 0.053355 0.037191 0.007214 0.020568 0.020437
0.035877 0.045796 0.069609 0.071516 0.066383 0.054909 0.037352 0.007885
0.020697 0.020851 0.035896 0.045901 0.069609 0.071516 0.066383 0.054946
0.037390 0.008062 0.021118 0.021304 0.036060 0.045901 0.069619 0.071534
0.066388 0.054946 0.037390 0.008062

0.000000 0.128505 0.178314 0.187154 0.183469 0.160480 0.111180 0.090787
0.059823 0.038243 0.048043 0.128505 0.178314 0.187154 0.183469 0.160480
0.111180 0.090787 0.059823 0.038247 0.048043 0.128514 0.178332 0.187170
0.183469 0.160526 0.111261 0.090928 0.059881 0.038310 0.048144 0.128602
0.178496 0.187377 0.184589 0.161360 0.112292 0.091174 0.060054 0.038878
0.048582 0.128728 0.178634 0.187378 0.184589 0.161360 0.112292 0.091630
0.060703 0.040293 0.049975 0.139709 0.184752 0.188268 0.184808 0.161760
0.112414 0.091637 0.060716 0.040295

0.000000 0.140764 0.171657 0.184366 0.180301 0.160796 0.144143 0.094425
0.073872 0.034107 0.050892 0.140764 0.171657 0.184366 0.180301 0.160796
0.144143 0.094425 0.073872 0.034107 0.050903 0.140791 0.171703 0.184385
0.180301 0.160853 0.144329 0.094501 0.073911 0.034159 0.051026 0.140853
0.171823 0.185440 0.181606 0.161755 0.144807 0.095648 0.075274 0.035310
0.051774 0.140958 0.172098 0.185567 0.181720 0.162626 0.145326 0.104534
0.090163 0.055429 0.055113 0.144102 0.173994 0.186744 0.182133 0.162684
0.145385 0.105021 0.090206 0.055451

0.055451 0.124894 0.133494 0.174311 0.189647 0.170413 0.136814 0.078218
0.058205 0.028983 0.032126 0.124894 0.133494 0.174311 0.189647 0.170413
0.136814 0.078218 0.058205 0.028985 0.032138 0.124927 0.133566 0.174338
0.189677 0.170500 0.136995 0.078326 0.058315 0.029075 0.032381 0.125046
0.133760 0.175678 0.191456 0.172373 0.138087 0.078748 0.060046 0.031166
0.033522 0.125861 0.134398 0.175896 0.192218 0.173957 0.151243 0.102475
0.085342 0.063468 0.067472 0.158959 0.161036 0.192567 0.192968 0.174087
0.151317 0.103293 0.085915 0.063849

0.063849 0.089605 0.099125 0.151874 0.153545 0.171295 0.119203 0.081311
0.034687 0.016452 0.041184 0.089605 0.099125 0.151874 0.153545 0.171295
0.119203 0.081311 0.034687 0.016452 0.041194 0.089639 0.099216 0.151975
0.153646 0.171363 0.119311 0.081399 0.034811 0.016681 0.041315 0.089698
0.099893 0.152575 0.155003 0.173385 0.121281 0.082099 0.038341 0.018959
0.042862 0.090982 0.100031 0.153155 0.157773 0.194173 0.126090 0.103422
0.087879 0.061593 0.079755 0.126793 0.133556 0.174598 0.168289 0.194348
0.126221 0.104715 0.089285 0.062824

0.000000 0.050554 0.060769 0.096649 0.162990 0.147307 0.155325 0.108597
0.036442 0.019273 0.029080 0.050554 0.060769 0.096649 0.162990 0.147307
0.155325 0.108597 0.036442 0.019274 0.029097 0.050595 0.060874 0.096852
0.163126 0.147381 0.155374 0.108694 0.036624 0.019741 0.029108 0.050595
0.060875 0.096852 0.166342 0.152923 0.167025 0.119680 0.047138 0.025923
0.030742 0.050858 0.061037 0.098967 0.197857 0.185960 0.200736 0.177100
0.105375 0.077186 0.072076 0.087498 0.095651 0.102021 0.198907 0.186483
0.200989 0.178591 0.106959 0.078820

0.000000 0.128505 0.178314 0.187154 0.183469 0.160480 0.111180 0.090787
0.059823 0.038243 0.048043 0.128505 0.178314 0.187154 0.183469 0.160480
0.111180 0.090787 0.059823 0.038247 0.048043 0.128514 0.178332 0.187170
0.183469 0.160526 0.111261 0.090928 0.059881 0.038310 0.048144 0.128602
0.178496 0.187377 0.184589 0.161360 0.112292 0.091174 0.060054 0.038878
0.048582 0.128728 0.178634 0.187378 0.184589 0.161360 0.112292 0.091630
0.060703 0.040293 0.049975 0.139709 0.184752 0.188268 0.184808 0.161760
0.112414 0.091637 0.060716 0.040295

0.000000 0.140764 0.171657 0.184366 0.180301 0.160796 0.144143 0.094425
0.073872 0.034107 0.050892 0.140764 0.171657 0.184366 0.180301 0.160796
0.144143 0.094425 0.073872 0.034107 0.050903 0.140791 0.171703 0.184385
0.180301 0.160853 0.144329 0.094501 0.073911 0.034159 0.051026 0.140853
0.171823 0.185440 0.181606 0.161755 0.144807 0.095648 0.075274 0.035310
0.051774 0.140958 0.172098 0.185567 0.181720 0.162626 0.145326 0.104534
0.090163 0.055429 0.055113 0.144102 0.173994 0.186744 0.182133 0.162684
0.145385 0.105021 0.090206 0.055451

0.055451 0.124894 0.133494 0.174311 0.189647 0.170413 0.136814 0.078218
0.058205 0.028983 0.032126 0.124894 0.133494 0.174311 0.189647 0.170413
0.136814 0.078218 0.058205 0.028985 0.032138 0.124927 0.133566 0.174338
0.189677 0.170500 0.136995 0.078326 0.058315 0.029075 0.032381 0.125046
0.133760 0.175678 0.191456 0.172373 0.138087 0.078748 0.060046 0.031166
0.033522 0.125861 0.134398 0.175896 0.192218 0.173957 0.151243 0.102475
0.085342 0.063468 0.067472 0.158959 0.161036 0.192567 0.192968 0.174087
0.151317 0.103293 0.085915 0.063849

0.063849 0.089605 0.099125 0.151874 0.153545 0.171295 0.119203 0.081311
0.034687 0.016452 0.041184 0.089605 0.099125 0.151874 0.153545 0.171295
0.119203 0.081311 0.034687 0.016452 0.041194 0.089639 0.099216 0.151975
0.153646 0.171363 0.119311 0.081399 0.034811 0.016681 0.041315 0.089698
0.099893 0.152575 0.155003 0.173385 0.121281 0.082099 0.038341 0.018959
0.042862 0.090982 0.100031 0.153155 0.157773 0.194173 0.126090 0.103422
0.087879 0.061593 0.079755 0.126793 0.133556 0.174598 0.168289 0.194348
0.126221 0.104715 0.089285 0.062824

0.000000 0.050554 0.060769 0.096649 0.162990 0.147307 0.155325 0.108597
0.036442 0.019273 0.029080 0.050554 0.060769 0.096649 0.162990 0.147307
0.155325 0.108597 0.036442 0.019274 0.029097 0.050595 0.060874 0.096852
0.163126 0.147381 0.155374 0.108694 0.036624 0.019741 0.029108 0.050595
0.060875 0.096852 0.166342 0.152923 0.167025 0.119680 0.047138 0.025923
0.030742 0.050858 0.061037 0.098967 0.197857 0.185960 0.200736 0.177100
0.105375 0.077186 0.072076 0.087498 0.095651 0.102021 0.198907 0.186483
0.200989 0.178591 0.106959 0.078820

0.000000 0.031002 0.093305 0.085795 0.112160 0.233943 0.190266 0.125467
0.086422 0.031071 0.020523 0.031002 0.093305 0.085795 0.112160 0.233943
0.190266 0.125467 0.086422 0.031071 0.020540 0.031310 0.093474 0.085988
0.112245 0.234005 0.190362 0.125608 0.086516 0.031087 0.020540 0.031310
0.093474 0.086002 0.119292 0.249645 0.211239 0.151128 0.108303 0.049731
0.028496 0.031885 0.093762 0.087494 0.161705 0.286236 0.247868 0.189686
0.175708 0.105676 0.066388 0.068014 0.129229 0.108173 0.162992 0.286616
0.248051 0.190621 0.176208 0.106599

0.106599 0.023272 0.034040 0.078579 0.104561 0.103848 0.091061 0.083988
0.076041 0.060215 0.010815 0.023272 0.034040 0.078579 0.104561 0.103848
0.091061 0.083988 0.076041 0.060215 0.011054 0.023768 0.034207 0.080103
0.104609 0.103967 0.091331 0.084095 0.076168 0.060215 0.011054 0.023768
0.034207 0.084265 0.120884 0.129146 0.124864 0.118300 0.111812 0.088513
0.026512 0.024997 0.034360 0.084460 0.126202 0.176799 0.165853 0.156333
0.171793 0.126444 0.060578 0.058192 0.061962 0.087260 0.126724 0.177847
0.167075 0.157888 0.172857 0.126538

0.126538 0.013118 0.028557 0.084757 0.135099 0.088132 0.080472 0.079324
0.049617 0.057847 0.018588 0.013118 0.028557 0.084757 0.135099 0.088132
0.080472 0.079324 0.049617 0.057851 0.018644 0.013902 0.028725 0.084980
0.135952 0.089921 0.080706 0.079702 0.049617 0.057851 0.018644 0.013902
0.029001 0.092446 0.155197 0.123206 0.116710 0.116875 0.086617 0.092593
0.041215 0.028283 0.029199 0.092486 0.155257 0.133727 0.165289 0.154661
0.136201 0.132680 0.074098 0.050353 0.046957 0.101128 0.155453 0.135035
0.166705 0.156599 0.138162 0.133725

0.000000 0.015259 0.025405 0.057238 0.077264 0.077775 0.086000 0.050877
0.043252 0.008833 0.005397 0.015259 0.025405 0.057238 0.077264 0.077775
0.086000 0.050877 0.043255 0.008846 0.005446 0.015841 0.025522 0.057474
0.077394 0.077972 0.086143 0.051066 0.043255 0.008846 0.005446 0.015841
0.026942 0.065499 0.102799 0.111010 0.123907 0.088142 0.080180 0.044463
0.032247 0.017881 0.030886 0.065526 0.102849 0.111035 0.135543 0.139015
0.115101 0.069569 0.046775 0.030179 0.031755 0.066304 0.102913 0.111532
0.135930 0.139770 0.116197 0.070718

0.070718 0.024567 0.036060 0.045901 0.069619 0.071534 0.066388 0.054946
0.037390 0.008062 0.021118 0.024567 0.036060 0.045901 0.069619 0.071534
0.066388 0.054946 0.037399 0.008103 0.021185 0.025188 0.036217 0.048039
0.070593 0.071679 0.066545 0.055177 0.037399 0.008103 0.021185 0.025188
0.036217 0.056068 0.090974 0.105671 0.103295 0.092529 0.074999 0.014833
0.044657 0.027209 0.041229 0.056134 0.091043 0.105713 0.103315 0.098465
0.099828 0.018908 0.049272 0.027749 0.042471 0.056170 0.091043 0.105713
0.103315 0.098465 0.100126 0.019830

0.000000 0.139709 0.184752 0.188268 0.184808 0.161760 0.112414 0.091637
0.060716 0.040295 0.049979 0.139709 0.184752 0.188268 0.184808 0.161760
0.112414 0.091637 0.060724 0.040331 0.050031 0.139768 0.185885 0.188466
0.184886 0.161932 0.112520 0.091828 0.060724 0.040331 0.050031 0.139768
0.185885 0.192661 0.199693 0.188126 0.146766 0.127541 0.096489 0.072710
0.069377 0.140991 0.186069 0.192760 0.200181 0.188154 0.146856 0.130148
0.109018 0.073762 0.069494 0.141052 0.186069 0.192760 0.200181 0.188154
0.146856 0.130148 0.109018 0.074167

0.000000 0.144102 0.173994 0.186744 0.182133 0.162684 0.145385 0.105021
0.090206 0.055451 0.055136 0.144102 0.173994 0.186744 0.182133 0.162684
0.145385 0.105021 0.090212 0.055488 0.055209 0.144423 0.174540 0.189496
0.182459 0.163212 0.145723 0.105272 0.090212 0.055488 0.055209 0.144423
0.174540 0.191818 0.190233 0.180591 0.166442 0.132360 0.121665 0.060357
0.056942 0.144649 0.174739 0.192754 0.190326 0.180605 0.166442 0.132460
0.121986 0.060796 0.057014 0.144649 0.174739 0.192754 0.190326 0.180605
0.166442 0.132460 0.121986 0.060796

0.000000 0.158959 0.161036 0.192567 0.192968 0.174087 0.151317 0.103293
0.085915 0.063849 0.067798 0.158959 0.161036 0.192567 0.192968 0.174087
0.151317 0.103293 0.085921 0.063890 0.067873 0.161062 0.165771 0.201739
0.204946 0.185276 0.152326 0.104707 0.085921 0.063890 0.067873 0.161062
0.165771 0.201800 0.205216 0.190694 0.162218 0.106746 0.101512 0.074500
0.069038 0.161147 0.167737 0.201933 0.205283 0.190708 0.162218 0.106746
0.101549 0.074704 0.069046 0.161147 0.167737 0.201933 0.205283 0.190708
0.162218 0.106746 0.101549 0.074704

0.000001 0.126793 0.133556 0.174598 0.168289 0.194348 0.126221 0.104715
0.089285 0.062824 0.080694 0.126793 0.133556 0.174598 0.168289 0.194348
0.126221 0.104715 0.089290 0.062849 0.080731 0.131140 0.145080 0.195873
0.193692 0.218867 0.129024 0.113841 0.094097 0.063435 0.080731 0.131140
0.145080 0.195873 0.193692 0.219593 0.129719 0.114618 0.094470 0.063765
0.080890 0.131271 0.145164 0.195937 0.193722 0.219593 0.129719 0.114638
0.094553 0.063765 0.080890 0.131271 0.145164 0.195937 0.193722 0.219593
0.129719 0.114638 0.094553 0.063765

0.000000 0.087498 0.095651 0.102021 0.198907 0.186483 0.200989 0.178591
0.106959 0.078820 0.072310 0.087498 0.095651 0.102021 0.198907 0.186483
0.200989 0.178591 0.106960 0.078827 0.074611 0.097926 0.119055 0.132910
0.233182 0.222195 0.204930 0.182597 0.114702 0.080378 0.074611 0.097926
0.119055 0.132910 0.233182 0.222356 0.205708 0.183887 0.114899 0.081128
0.075752 0.098185 0.119312 0.132988 0.233210 0.222360 0.205730 0.183887
0.114972 0.081318 0.075757 0.098185 0.119312 0.132988 0.233210 0.222360
0.205730 0.183887 0.114972 0.081318

0.000000 0.068014 0.129229 0.108173 0.162992 0.286616 0.248051 0.190621
0.176208 0.106599 0.066601 0.068014 0.129229 0.108173 0.162992 0.286616
0.248051 0.190621 0.176208 0.107542 0.070766 0.083223 0.157788 0.142298
0.199701 0.325753 0.253098 0.195019 0.190810 0.109497 0.071444 0.083529
0.158170 0.142609 0.200165 0.326739 0.254552 0.196198 0.190908 0.109733
0.071671 0.083804 0.158407 0.142713 0.200212 0.326765 0.254651 0.196198
0.190908 0.109782 0.071671 0.083804 0.158407 0.142713 0.200212 0.326765
0.254651 0.196198 0.190908 0.109782

0.000000 0.058192 0.061962 0.087260 0.126724 0.177847 0.167075 0.157888
0.172857 0.126538 0.060728 0.058192 0.061962 0.087260 0.126724 0.177847
0.167075 0.157888 0.172857 0.126538 0.066673 0.075531 0.091114 0.123350
0.163657 0.217404 0.172067 0.184240 0.186152 0.132903 0.068504 0.076853
0.092280 0.124453 0.165032 0.219513 0.173767 0.184389 0.186403 0.135049
0.068754 0.077170 0.092597 0.124622 0.165115 0.219593 0.173885 0.184389
0.186403 0.135049 0.068754 0.077170 0.092597 0.124622 0.165115 0.219593
0.173885 0.184389 0.186403 0.135049

0.000000 0.050353 0.046957 0.101128 0.155453 0.135035 0.166705 0.156599
0.138162 0.133725 0.074434 0.050353 0.046957 0.101128 0.155453 0.135035
0.166705 0.156599 0.138162 0.133725 0.076810 0.066875 0.074410 0.135521
0.192525 0.172302 0.171228 0.181058 0.150824 0.137905 0.078895 0.068217
0.075507 0.136365 0.193682 0.173548 0.172129 0.181220 0.151089 0.138206
0.079171 0.068901 0.075748 0.136459 0.193705 0.173553 0.172129 0.181220
0.151089 0.138206 0.079171 0.068901 0.075748 0.136459 0.193705 0.173553
0.172129 0.181220 0.151089 0.138206

0.138206 0.030179 0.031755 0.066304 0.102913 0.111532 0.135930 0.139770
0.116197 0.070718 0.047585 0.030179 0.031755 0.066304 0.102913 0.111532
0.135930 0.139770 0.116197 0.070718 0.049797 0.039195 0.053889 0.097568
0.137946 0.146238 0.163959 0.161490 0.124380 0.072223 0.050610 0.040171
0.054877 0.098156 0.138637 0.146752 0.164624 0.163051 0.124561 0.073116
0.051210 0.040335 0.055034 0.098221 0.138646 0.146752 0.164624 0.163051
0.124561 0.073116 0.051210 0.040335 0.055034 0.098221 0.138646 0.146752
0.164624 0.163051 0.124561 0.073522

0.073522 0.027749 0.042471 0.056170 0.091043 0.105713 0.103315 0.098465
0.100126 0.019830 0.050543 0.027749 0.042471 0.056170 0.091043 0.105713
0.103315 0.098465 0.100126 0.019830 0.051697 0.033674 0.054250 0.076674
0.116309 0.108810 0.122090 0.109019 0.103749 0.021322 0.053206 0.035177
0.056358 0.078152 0.117612 0.110205 0.123631 0.109298 0.104989 0.022231
0.053436 0.035348 0.056565 0.078162 0.117612 0.110205 0.123631 0.109298
0.104989 0.022231 0.053436 0.035431 0.056605 0.078162 0.117612 0.110205
0.123631 0.109298 0.105286 0.023150

0.095740 0.089848 0.070838 0.050209 0.018396 0.003389 0.003389 0.003389
0.017086 0.035827 0.098392 0.122725 0.108403 0.088621 0.057121 0.008801
0.022183 0.022183 0.020133 0.039454 0.103325 0.141772 0.137085 0.123017
0.093501 0.044051 0.051729 0.026937 0.021189 0.039495 0.103325 0.141772
0.137134 0.123059 0.093502 0.044051 0.051729 0.026937 0.021189 0.039495
0.103325 0.141821 0.137982 0.123279 0.093768 0.044177 0.051987 0.027380
0.021291 0.039613 0.103557 0.142056 0.138078 0.123279 0.093768 0.044177
0.051987 0.027380 0.021332 0.039691 0.00 0.00 0.00

APPENDIX B

The 10x10 cost values:

1.018643 0.622603 1.223525 1.352929 1.360540 2.568414 1.323577 2.765275 1.609030 2.018555

2.699680 2.035996 0.783962 0.519023 0.431838 1.071347 0.381317 1.033432 0.703287 0.646479

2.865651 1.347955 3.175738 1.783549 2.527956 2.816140 1.716187 3.206208 1.804267 2.729310

1.368721 0.403301 1.410983 0.945442 1.126580 1.330500 0.834105 1.907370 1.255636 2.289300

2.871589 1.910008 3.376514 1.909288 2.781388 2.900722 2.015819 4.102083 2.264710 3.335683

2.106268 1.524660 2.603455 1.488128 2.292374 2.129359 1.540216 2.627110 1.524368 2.313778

3.489862 2.263908 4.167512 2.283026 3.338918 3.592025 2.427873 4.705108 3.291558 4.379825

2.357738 1.561198 2.635066 1.536828 2.320528 2.454726 2.375507 3.599548 2.533881 3.858682

3.952441 3.398223 4.921279 3.508700 4.664248 4.758035 3.638731 5.447157 3.619450 4.829804

2.206886 2.684469 2.933323 2.609965 3.726703 3.095605 3.066571 3.733389 2.695374 3.753008

BIBLIOGRAPHY

BIBLIOGRAPHY

1. DARPA, "Neural Network Study," AFCEA International Press, VA, 1988.

2. Hopfield, J.J., "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proceeding of National Academic Science U.S.A.*, Vol. 79, pp. 2554-2558, 1982.

3. Rumelhart, D.E., Hinton, G.E., and Williams, R.J., "Learning Internal Representations by Error Propagation," *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, Cambridge, MA: MIT Press, 1986, Vol. I, pp. 318-362.

4. Grossberg, S., "Neural Networks and Natural Intelligence," MIT Press, Cambridge MA, 1988.

5. Grossberg, S., "Adaptive Pattern Classification and Universal Recoding: I. Parallel Development and Coding of Neural Feature Detectors," *Biological Cybernetics*, Vol 23, pp. 121 - 134, 1976.

6. Kohonen, T., "Self-Organization and Associative Memory," Second Edition, Springer-Verlag, Berlin, 1988.

7. Hopfield, J. J., "Artificial Neural Networks," *IEEE Circuits and Device Magazine*, pp. 3-10, Sep. 1988.

8. Vemuri, V., "Artificial Neural Networks: An Introduction," *Artificial Neural Networks: Theoretical Concepts*, IEEE Computer Society Press Technology Series, pp. 1-12, 1988.

9. Hebb, D., "The Organization of Behavior," Wiley, New York, 1949.

10. Rosenblatt, F., "Principles of Neurodynamics," Spartan Books, Washington DC, 1961.

11. Lippman, R. P., "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, pp. 4 - 22, April, 1987.

12. Kohonen, T., "An Introduction to Neural Computing," *Neural Networks*, Vol. 1, No. 1, pp. 3- 16, 1988.

13. Kohonen, T., "Correlation Matrix Memories," *IEEE Transactions on Computers*, Vol. C-21, No. 4, pp. 353-359, 1972.

14. Hopfield, J.J., "Neurons with Graded Response Have Collective Computational Properties Like Those of Two-state Neurons," *Proceedings of the National Academy of Science USA*, Vol. 81, pp. 3088-3092, May 1984.

15. Hopfield, J.J. and Tank, D.W., "'Neural' Computation of Decisions in Optimization Problems," *Biological Cybernetics*, Vol. 52, pp. 141-152, 1985.

16. Rumelhart, D.E., Hinton, G.E., and Williams, R.J., "Learning Representations by Back-Propagation Errors," *Nature*, Vol. 323, pp. 533-536, 1986.

17. Minsky, M., and Papert, S., "Perceptrons," MIT Press, Cambridge MA, 1969.

18. Hebb, D., "The Organization of Behavior," Wiley, New York, 1949.

19. Rosenblatt, F., "Principles of Neurodynamics," Spartan Books, Washington DC, 1961.

20. Tank, D.W. and Hopfield, J.J., "Simple Neural Optimization Networks: An A/D Converter, Signal Decision Network, and a Linear Programming Circuit," *IEEE Transactions on Circuits and Systems*, Vol. CAS-33, No. 5, pp. 533-541, 1986.

21. Kahng, A.B., "Traveling Salesman Heuristics and Embedding Dimension in the Hopfield Model," *Proceedings of the IEEE International Conference on Neural Networks*, Vol. I, pp. 513-520, 1989.

22. Wilson, G.V. and Pawley, G.S., "On the Stability of the TSP Problem Algorithm of Hopfield and Tank", *Biological Cybernetics* 58, pp. 63-70, 1988.

23. Angeniol, B. *et al.*, "Self-Organizing Feature Maps and the Travelling Salesman Problem," *Neural Networks*, Vol. 1, pp. 289-293, 1988.

24. Maa, C.-Y., and Shanblatt, M. A., "Stability of Linear Programming Neural Network for Problems with Hypercube Feasible Region," *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, CA, June 1990.

25. Maa, C.-Y., and Shanblatt, M. A., "Improved Linear Programming Neural Networks," *Proceedings of 32nd Midwest Symposium on Circuits and Systems*, Champaign, IL, pp. 740 - 743, August, 1989.

26. Kennedy, M.P. and Chua, L.O., "Neural Networks for Nonlinear Programming," *IEEE Transactions on Circuits and Systems*, Vol. CAS-35, No. 5, pp. 554-562, May 1988.

27. Maa, C.-Y., and Shanblatt, M. A., "Neural Networks for Nonlinear Programming," in review, *IEEE Transactions on Neural Networks*.

28. Rauch, H.E. and Winarske, T., "Neural Networks for Routing Communication Traffic," *IEEE Control Systems Magazine*, pp. 26-31, April, 1988.

29. Zhang, L. and Thomopoulos, S. C. A., "Neural Network Implementation of the Shortest Path Algorithm for Traffic Routing in Communication Networks," *Proceedings of the IEEE International Conference on Neural Networks*, Washington, D. C., Vol. 2, p. 591, June 1989.

30. Jeffery, W. and Rosner, R., "Optimization Algorithms: Simulated Annealing and Neural Network Processing," *Astrophysical Journal*, Vol. 310, Nov. 1986, pp. 473-481.

31. Kamgar-Parsi, B. and Kamgar-Parsi, B., "An Efficient Model of Neural Networks for Optimization," *Proceedings of the IEEE International Conference on Neural Networks*, Vol. III, pp. 785-790, 1987.

32. Levy, B.C. and Adams, M.B., "Global Optimization with Stochastic Neural Networks," *Proceedings of the IEEE International Conference on Neural Networks*, Vol. III, pp. 681-689, 1987.

33. Ramanujam, J. and Sadayappan, P., "Optimization by Neural Networks," *Proceedings of the IEEE International Conference on Neural Networks*, Vol. II, pp. 325-332, 1988.

34. Chiu, C., Maa, C.-Y., and Shanblatt, M.A., "An Artificial Neural Network Algorithm for Dynamic Programming,", *International Journal of Neural Systems*, Vol 1, No. 3, pp. 211-220, 1990.

35. Chiu, C., Maa, C.-Y., and Shanblatt, M.A., "Energy Function Analysis for Dynamic Programming Neural Networks," to appear, *IEEE Transactions on Neural Networks*, Vol. 2, No. 4, July, 1991.

36. Kirk, D.E., "Optimal Control Theory: An Introduction," Englewood Cliffs. N.J., Prentice-Hall, pp. 53-78, 1970.

37. Salam, F. M. A., Wang, Y., and Choi, M. R., "On the Analysis of Dynamic Feedback Neural Nets," *IEEE Transactions on Circuits and Systems*, Vol. 38, No. 2, pp. 196 - 201, Feb. 1991.

38. Kosko, B., "Bidirectional associative memories," *IEEE Transactions on System, Man, and Cybernetics*, Vol. 18, no. 1, pp. 49-60, Jan./Feb. 1988.

39. Wang, Y.-F., *et. al.*, "Two Coding Strategies for Bidirectional Associative Memory," *IEEE Transactions on Neural Networks*, Vol. 1, no. 1, March 1990.

40. Li, J. H., Michel, A. N., and Porod, W., "Qualitative Analysis and Synthesis of a Class of Neural Networks," *IEEE Transactions on Circuits and Systems*, Vol. 35, No. 8, pp. 976-986, Aug. 1988.

41. Farrel, J.A., and Michel, A.N., "A Synthesis Procedure for Hopfield's Continuous-Time Associative Memory," *IEEE Transactions on Circuits and Systems*, Vol. 37, No. 7, pp.877-884, July 1990.

42. Sakoe, H., and Chiba, S., "Dynamic Programming Algorithms Optimization for Spoken Word Recognition," *IEEE Transactions on ASSP*, Vol. 26, No. 1, pp. 43 - 49, Feb. 1978.

43. Itakura, F., "Minimum Prediction Residual Principle Applied to Speech Recognition," *IEEE Transactions on ASSP*, Vol. 23, No. 1, pp. 67-72, Feb. 1975.

44. Myers, C., Rabiner, L. R., and Rosenberg, A. E., "Performance Trade-off in Dynamic Time Warping Algorithms for Isolated Word Recognition," *IEEE Transactions on ASSP*, Vol. 28, No. 6, pp. 623-635, Dec. 1980.

45. White, G. M., and Neely, R. B., "Speech Recognition Experiments with Linear Prediction, Bandpass Filtering and Dynamic Programming," *IEEE Transactions on ASSP*, Vol. 24, pp. 183-188, Apr. 1976.

46. Chiu, C., and Shanblatt, M. A., "A Neural Network Method for Dynamic Time Warping on Speech Recognition,", in review, *IEEE Transactions on ASSP*.

47. Eberhardt, S., Duong, T. and Thakoor, A., "Design of Parallel Hardware Neural Network Systems From Custom Analog VLSI "Building Block" Chips," *Proceedings*

of the IEEE International Conference on Neural Networks, Vol. 2 pp. 183-190, 1989.

48. Mead, C., "Analog VLSI and Neural Systems," Addison-Wesley, 1989.

49. Hecht-Nielsen, R., "Neurocomputing," Addison-Wesley, 1989.

50. McCulloch, W.S., and Pitts, W.H., "A Logical Calculus for the Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, Vol. 5, pp. 115-133, 1943.

51. Hoppensteadt, F.C., "An Introduction to the Mathematics of Neurons," Cambridge University Press, 1986.

52. LeCun, Y., *et al.*, "Backpropagation Applied to Handwritten Zip Code Recognition," Neural Computation, Vol. 1, pp. 541-551, 1989.

53. Weideman, W. E., "A Comparison of a Nearest Neighbor Classifier and a Neural Network for Numeric Handprint Character Recognition," *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, Vol. I, pp. 117-120, July 1988.

54. Kammerer, B. R., and Kupper, W. A., "Experiments for Isolated-Word Recognition with Single- and Two-layer Perceptrons," *Neural Networks*, Vol. 3, pp. 693-706, 1990.

55. Waibel, A., et al., "Phoneme Recognition Using Time-Delay Neural Networks," *IEEE Trans. on ASSP*, Vol 37, No. 3, March 1989.

56. Irino, T., and Kawahara, H., "A Method for Designing Neural Networks Using Nonlinear Multivariate Analysis: Application to Speaker-Independent Vowel Recognition," *Neural Computation*, Vol. 2 386 - 397, 1990.

57. Guez, A., Ahmad, Z., "Solution to the Inverse Kinematics Problem in Robotics by Neural Networks," *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, Vol. II pp. 617-624, July 1988.

58. Hosogi, S., "Manipulator Control Using Layered Neural Network Model with Self-Organizing Mechanism," *Proceedings of the IEEE International Conference on Neural Networks*, Washington, DC, Vol II, pp. 217 - 220, Jan. 1990.

59. Nguyen, D., and Widrow, B., "The Truck Back-Upper: An Example of Self-Learning in Neural Networks," *Proceedings of the IEEE International Conference on Neural*

Networks, Washington, DC, Vol. II, pp. 357 - 363, June 1989.

60. Kuperstein, M., and Wang, J., "Neural Controller for Adaptive Movements with Unforeseen Payloads," *IEEE Transactions on Neural Networks*, Vol., 1 No.1, March 1990.

61. Tsirukis, A. G., *et al.*, "Nonlinear Optimization Using Generalized Hopfield Networks," *Neural Computation*, Vol 1, No. 4, pp. 511 - 520, 1989.

62. Protzel, P. W., "Comparative Performance Measure for Neural Networks Solving Optimization Problems," *Proceedings of the IEEE International Conference on Neural Networks*, Washington, DC, Vol II, pp. 523 - 526, Jan. 1990.

63. Li, W., and Nasrabadi, M., "Object Recognition Based on Graph Matching Implemented by a Hopfield-Style Neural Network," *Proceedings of the IEEE International Conference on Neural Networks*, Washington, DC, Vol II, pp. 287 - 290, June 1989.

64. Parvin, B., and Medioni, "A Constraint Satisfaction Network for Matching 3D Objects," *Proceedings of the IEEE International Conference on Neural Networks*, Washington, DC, Vol II, pp. 281 - 286, June 1989.

65. Mjolsness, E., Gindi, G., and Anandan, P., "Optimization in Model Matching and Perceptual Organization," *Neural Computation*, Vol. 1, pp. 218 -229, 1989.

66. Rosenbaltt, F., "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review*, Vol. 65, pp. 385 - 408, 1958.

67. Widrow, B., and Hoff, M. E., "Adaptive Switching Circuits," *WESCON Convention Record*, Vol. 4, pp. 96 - 104, 1960.

68. Weideman, W. E., "A Comparison of a Nearest Neighbor Classifier and a Neural Network for Numeric Handprint Character Recognition," *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, Vol. I, pp. 117-120, July 1988.

69. Elsley, R.K., "A Learning Architecture for Control Based on BackPropagation Neural Networks," *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, Vol. II pp. 587-594, July 1988.

70. Graf, H.P., *et. al.*, "VLSI Implementation of a Neural Network Memory with Several Hundred Neurons," *Proc. of AIP Conf. on Neural Networks* for Computing, No. 151, pp. 182-187,1986.

71. Graf, H.P., *et. al.*, "A CMOS Association Memory Chip," *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, Vol. III, pp. 461-468, 1987.

72. Graf, H.P., *et. al.*, "VLSI Implementation of a Neural Network Model," *Computer*, Vol. 21, No. 3, pp. 41-49, March 1988.

73. Sivilotti, M.A., Ashowald, M.A., and Mead, C.V., "Real-Time Visual Computation Using Analog CMOS Processing Array," *Advanced Research in VLSI: Proceedings of the 1987 Stanford Conference*, P. Losleben (Ed.), Cambridge, MA: MIT Press, pp. 295-312, 1987.

74. Sivilotti, M.A., Emerling, M.R., and Mead, C.V., "VLSI Architectures for Implementation of Neural Networks," *Proc. of AIP Conf. on Neural Network for Computing*, No. 151, pp. 408-413, 1986.

75. Vidyasagar, M., "Nonlinear Systems Analysis," Englewood Cliffs. N.J., Prentice-Hall, pp. 131-186, 1978.

76. Hirsch, M.W., and Small, S., "Differential Equations, Dynamical Systems, and Linear Algebra," New York, Academic Press, pp. 199-204, 1974.

77. Wilson, G.V. and Pawley, G.S., "On the Stability of the TSP Problem Algorithm of Hopfield and Tank", *Biological Cybernetics* 58, pp. 63-70, 1988.

78. Davis, G. W., Jr., "Sensitivity Analysis in Neural Net Solutions," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 19, No. 8, 1078 - 1082, Sep./Oct. 1989.

79. Aiyer, S.V.B., Niranjan, M., and Fallside, F. "A Theoretical Investigation into the Performance of the Hopfield Model," *IEEE Transaction on Neural Networks*, Vol. 1, No 2, pp. 204-215, June 1990.

80. Abe, S, "Theories on the Hopfield Networks," *Proceedings of the IEEE International Conference on Neural Networks*, Vol. I, pp. 557 - 563, July, 1989.

81. Ortega, J.M., "Matrix Theory," Plenum Press, New York, 1987.

82. Salam, F. M. A., Wang, Y., and Choi, M. R., "On the Analysis of Dynamic Feedback Neural Nets," *IEEE Transactions on Circuits and Systems*, Vol. 38, No. 2 pp. 196 - 201, Feb. 1991.

83. Eberhardt, S., Duong, T. and Thakoor, A., "Design of Parallel Hardware Neural Network Systems From Custom Analog VLSI "Building Block" Chips," *Proceedings of the IEEE International Conference on Neural Networks,* San Diego, Vol II, pp. 183-190 1989.

84. Agranat, A. J., Neugebauer, C. F., and Yariv, A., "A CCD Based Neural Integrated Circuit with 64 K Analog Programmable Synapses," *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, Vol II, pp. 551 - 555, June 1990.

85. Fisher, W. A., Fujimoto, R. J., and Okamura, M. M. "The Lockheed Programmable Analog Neural Network Processor," *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, Vol II, pp. 563 - 568, June 1990.

86. Lee, B. W., Lee, J.-C., and Sheu, B. J., "VLSI Image Processors Using Analog Programmable Synapses and Neurons," *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, Vol II, pp. 563 - 568, June 1990.

87. Swaminathan, G., *et al.*, "Fault Tolerance in Neural Networks," *Proceedings of the IEEE International Conference on Neural Networks*, Washington, D. C. Vol II, pp. 699 - 702, June 1989.

88. Silverman, H. F., and Morgan, D. P., "The Application of Dynamic Programming to Connected Speech Recognition," *IEEE ASSP Magazine*, pp. 7-25, July 1990.

89. Picone, J., "Continuous Speech Recognition Using Hidden Markov Models," *IEEE ASSP Magazine*, pp. 26-41, July 1990.

90. Itakura, F., "Minimum Prediction Residual Principle Applied to Speech Recognition," *IEEE Transaction on ASSP*, Vol. 23, No. 1, pp. 67-72, Feb. 1975.

91. Myers, C., Rabiner, L. R., and Rosenberg, A. E., "Performance Trade-off in Dynamic Time Warping Algorithms for Isolated Word Recognition," *IEEE Transaction on ASSP*, Vol. 28, No. 6, pp. 623-635, Dec. 1980.

92. White, G. M., and Neely, R. B., "Speech Recognition Experiments with Linear Predic-

tion, Bandpass Filtering and Dynamic Programming," *IEEE Transaction on ASSP*, Vol. 24, pp. 183-188, April 1976.

93. Kammerer, B. R., and Kupper, W. A., "Experiments for Isolated-Word Recognition with Single- and Two-layer Perceptrons," *Neural Networks*, Vol. 3, pp. 693-706, 1990.

94. Waibel, A., *et al.*, "Phoneme Recognition Using Time-Delay Neural Networks," *IEEE Transaction on ASSP*, Vol 37, No. 3, March 1989.

95. Rabiner, L. R., Rosenberg, A. E., and Levinson, S. E., "Considerations in Dynamic Time Warping Algorithms for Discrete Word Recognition," *IEEE Transaction on ASSP*, Vol. 26, No. 6, pp. 575 - 582, Dec. 1978.

96. Glinski, S., *et al.* "The Graph Search Machine (GSM): A Programmable Processor for Connected Word Speech Recognition and Other Applications," *Proceedings of ICASSP*, Dallas, TX, pp. 519-522, April 1987.

97. Mann, J. R., and Rhodes, F. M., "A Wafer DTW Multiprocessor," *Proceedings of ICASSP*, Tokyo, Japan, pp. 1557-1560, April 1986.

98. Jutand, F. C., *et al.*, "VLSI Architectures for Dynamic Time Warping Using Systolic Arrays," *Proceedings of ICASSP*, San Diego, CA, pp. 34A.5.1-34.A.5.4, March 1984.

99. Quenot, G., *et al.* "A Dynamic Time Warp VLSI Processor for Continuous Speech Recognition," *Proceedings of ICASSP*, Tokyo, Japan. pp. 1549-1552, April 1986.

100. Hwang, K., "Computer Arithmetic: Principle, Architecture, and Design," John Wiley and Sons, 1979.