Liapunov Functions as a Basis for Analysis of Neural-Network Models

by

Shelly Dawn Dalton Goggin

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science (Mathematics).

Golden, Colorado

Date August 22, 1988

Signed: Shelly Dawn Dalton Goggin

Shelly Dawn Dalton Goggin

Approved: Rodney Schmidt

Dr. Rodney Schmidt
Thesis Advisor

Golden, Colorado

Date August 22, 1988

Ardel Boes

Dr. Ardel Boes
Professor and Head,
Mathematics Department

ABSTRACT

Processing with neural-network models can provide solutions to many difficult problems in artificial intelligence, but different models are currently being analyzed in different ways. The functional components of different neural-network models tend to be similar, while the dynamics of the models vary drastically. Liapunov's method of stability analysis provides a method of analyzing these differences in model dynamics. The process of developing an appropriate Liapunov function also clarifies many of the assumptions and restrictions on model parameters necessary for optimal model performance. This paper demonstrates the dynamic stability of four representative neural-network models using Liapunov functions. The models are: the outer-product model, the Boltzman machine, adaptive resonance theory and the backpropagation model. A software simulation of the backpropagation model shows how a Liapunov function can also be used to choose parameters for an implementation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation, contract number NSF CDR 8622236.

I would like to thank Dr. Rod Schmidt for advising and supporting me for the past year. Thanks to Drs. Aaron Gordon and Robert Underwood for their advise and editing. I would like to acknowledge many useful discussions with Drs. Steve Pruess, Skip Vechia and Jean Bell, as well as Kathy Rossmiller and the students of the JOLY group at the University of Colorado. A special thanks to Dr. Kris Johnson, and of course, John Goggin, my family and friends, without whom I would have never made it.

Chapter 1
INTRODUCTION

1.1 Purpose and Scope

Neurobiologists, psychologists, computer scientists, mathematicians and engineers use neural-network models to study the functioning of the brain and other massively-parallel processing systems. Such research continues to produce a wide variety of neural-network models, each rooted in the concepts of the originating field or fields. For instance, while psychologists use the term "associative memory" to refer to the linkage of concepts in the mind, a computer scientist is much more likely to consider the term to denote a means of address-independent data retrieval. Even the preferred name for the research varies from discipline to discipline. Biologically influenced terms include artificial neural systems, perceptrons and neural networks, while the names adopted by engineers include connectionist systems and cybernetics. This disparity of concepts and terminology leads to a "Tower of Babel" effect as researchers are limited in their ability to contrast and evaluate models originating in fields other than their own. For example, the mathematical dissertation produced by Stephen Grossberg in 1968 presents a model which was later independently derived by John Hopfield in his 1979 studies in neurobiology, and, to this day, the extent to which the two models replicate each other is unclear[1] . One way to overcome this type of problem in analysis is the use of a consistent mathematical approach which emphasizes the salient features of the models. Analysis based on Liapunov's second method is the approach presented in this thesis.

The functional components of neural-network models tend to be similar, while the dynamics vary greatly. Nearly all neural-network models are made up of a system of units (or neurons) which process a set of weighted inputs. The method of processing within each of the units provides the dynamics for the model, the effect of which can range from error minimization to response generalization. However to perform useful computations, the model must have a determinable stability in its dynamics. If a researcher is to interpred a neural-network model, the model must generate a meaningful response within a reasonable amount of time. This requirement fo a timely response maps to a requirement for at least asymptotic stability, in the model's dynamics. Since Liapunov functions characterize stability, their use provides a useful approach to neural-network model analysis empahsizing model dynamics. Models with different dynamics have different forms of Liapunov functions and different stable regions of the parameter space.

Chapters 2-5 provide examples of stability analysis of several neural-network models using Liapunov functions. The models used in the analyses are representative of contemporary neural-network models from a variety of disciplines: outer-product model from biology, the Boltzman machine from statistical processing, adaptive resonance theory from psychology, and the backpropagation model from cognitive science. Although seldom described as such in the literature detailing these models, Liapunov functions are used under different names to describe the dynamics of each of these models.

Chapter 6 contains a parameter analysis for the stability of the backpropagation model. This model was chosen for parameter analysis because construction of a simulation is straightforward.

Descriptions of neural-network models are often incomplete and/or vague. Necessary assumptions about the models which are not clearly stated in the available literature are noted where appropriate. Where discussion pertains to a particular variant of the more general model under consideration, this, too, has been noted. Since the scope of this thesis does not include a comprehensive analysis of all facets of the models, this discussion is limited to analysis which pertains to the each model's dynamics as expressed by a Liapunov function.

The results of these analyses are pertinent both to the particular models treated as well as to neural-network models in general. For the individual models, assumptions and parameter restrictions required for optimal dynamics of each model are explored. Comparisons of the Liapunov functions make clear the differences among the dynamics of individual models. The approach presented here places emphasis on the behavior of the models as opposed to emphasizing the variations in the components of the models. Although functional components tend to attract the initial attention of researchers interested in neural-network models, ultimately an understanding of the dynamics is required before useful development and implementation can proceed.

1.2 Background on Neural-Network Models

The first obstacle to understanding neural-network models is the terminology used in the descriptions. In nearly every presentation of a neural-network model at least one change in the notation or terminology is found. Since no standard exists, the notation and the terminology used here is a combination of the terminology found in several models. The models analyzed in later chapters will be translated to this notation so that the similarities and differences can be readily seen. In addition to defining the terminology in this section, neural-network models are differentiated from mathematically similar vector-matrix multipliers, and a brief history of neural-network models is presented.

The first choice in terminology is the name of the models. The name neural-network models was chosen to emphasize both the biological and the systems engineering aspects of the models. The choice of the term model emphasizes the fact that both neurobiological research and massively parallel processing research are in the formative stages. Other terms, such as connectionist models, parallel distributed processing models and artificial neural systems were rejected here due to their close association with specific research groups.

The components of neural-network models are grouped into three major functional categories: connections between processing units, weights for inputs to the units, and a nonlinear decision function for determining the output of a unit. This categorization scheme is similar to that used in the book, *Parallel Distributed Processing[2]* . The connections between the processing units describe how the network of units is arranged and the direction of the propagation of signals, or data-transfer values, between units in the network. Each unit has a set of weights associated with the data-transfer values which indicate the relative importance of the different inputs to the state of the unit. The state of a unit is computed through the decision function which has two sub-functions: the activation function determines the activation level of the unit from the weighted input values, and the threshold function determines the data-transfer value of the unit from the activation level. Figure 1.1 shows a simplified diagram for a unit in a general neural-network model. Specific neural-network models contain variations in the details of the three functional components and the subcomponents.

Different neural-network models vary with respect to the direction of propagation of the data-transfer values through the connections from unit to unit. The direction (lateral, feed-forward or feedback) is determined relative to the external input vector. The external input vector is supplied by the user for processing. The external input vector is either a vector of input values for some or all of the units, or the initial data-transfer values of some or all of the units. The presentation of the external input vector marks the beginning of an event for the model. The reception of the output data transfer values by the user from some, or all of the units marks the end of an event. If the weights which represent the values of the connections between units are represented by a single matrix, then then the model is considered to be a single layer neural-network and usually each unit is connected to all of the other units in the model. In this case, the propagation of signals is lateral with respect to the layer. In multi-layer neural-network models, the units on

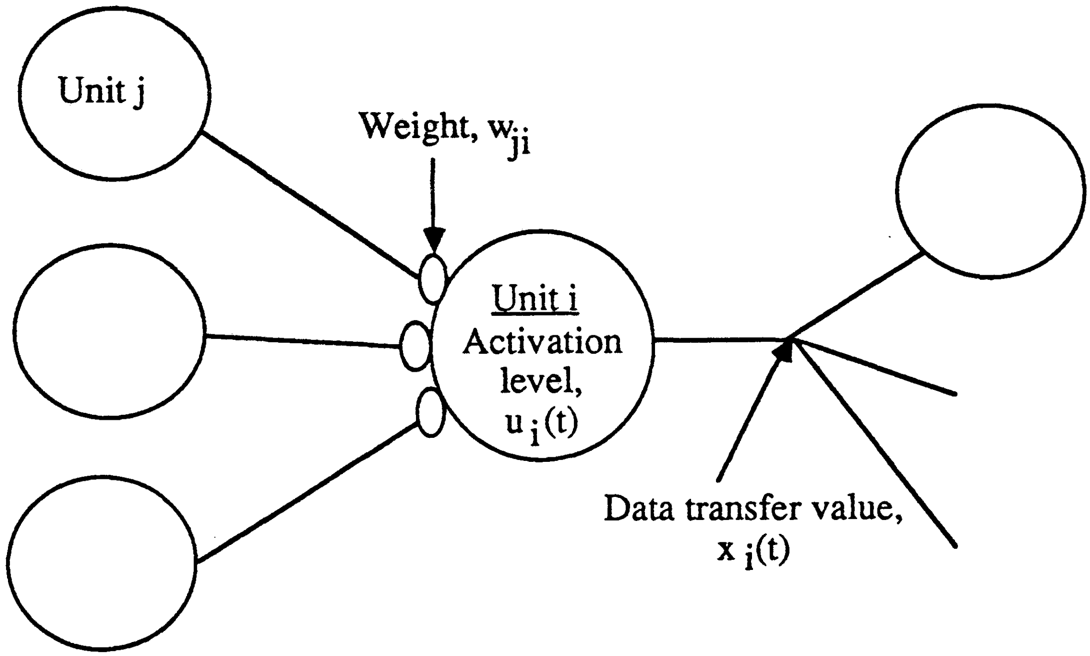


Figure 1.1 Basic Neural-Network Model's Components

different layers have different weight matrices. For instance, the units in the input layer have a weight matrix representing the connections from each unit in the layer to each element of the external input vector. The units in the output layer have a matrix representing the connections from the units in the proceeding layer to the units in the output layer. The data-transfer values from this layer are interpreted by the user as the output of the model. Each hidden layer has a matrix representing the connections which propagate data-transfer values between the input and output layers. When the propagation of the data-transfer values is from the input layer towards the output layer, the connections are feed-forward. Propagation of the data-transfer values from the output layer back towards the input layer is performed with feedback connections. As with the single layer model, lateral connections can exist between units in the same layer. Figure 1.2 is a schematic of these variations in connections.

Different neural-network models use different methods for determining the values of the weights. The values of the weights are either determined before the model is used, or the values of the weights are modified using information available during processing. The process of modifying the weights is referred to as adaptation of the weights. If no adaptation is performed, then the values of the weights representing the relative importance of the inputs to the units are determined a priori by the application. Thus, in pattern association applications, the weights are set to the sum of the outer product of the external input vector and the associated desired output vector. In optimization applications, the weights are set to values which reflect the appropriateness of high data-transfer values for both units connected by the weight. Unfortunately, many problems do not allow a priori calculation of the weights.

Adaptation is used to determine the values of the weights when the application does not clearly indicate the best values for the weights. For instance, if the application is to generalize the response to a set of external

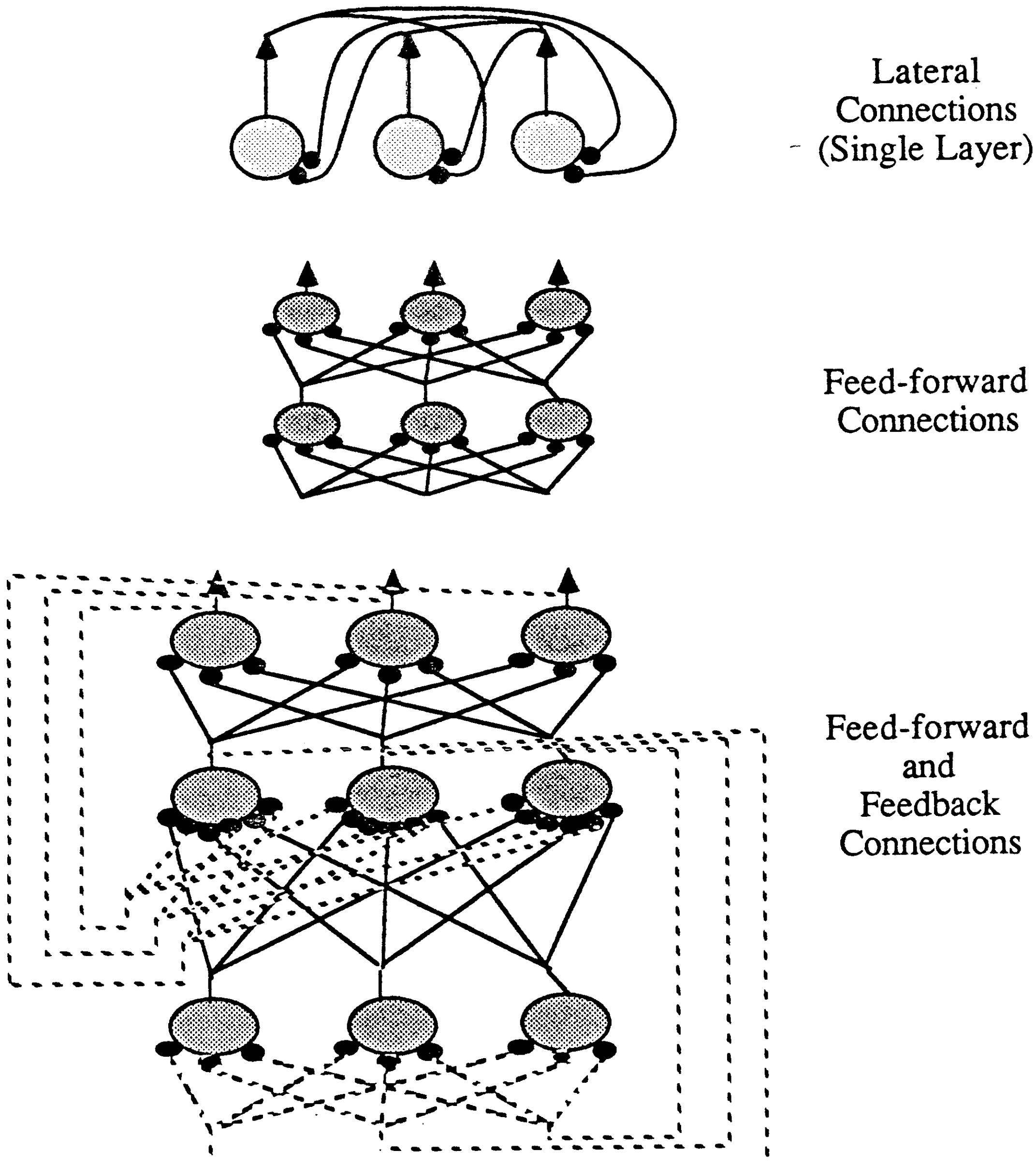


Figure 1.2 Connections Between Units in Neural-Network Models

input vectors, each unit representing a category adapts its weights to reflect the salient features of the input vectors which may not be recognized by the operator. For an association application, pairs of noisy external input vectors and desired output vectors are presented to the model and the weights are adapted to represent the sum of the outer products of the pairs of vectors without noise. Variations in the types of adaptation depend on the criteria used to change the values of the weights. In a model for generalization, the main criterion is a measure of similarity between external input vectors. In a neural-network model for recognizing noisy patterns, the main criterion is the error between the actual and desired output. The differences in criteria are visible in the form of the Liapunov functions which describe the dynamics of the adaptation. Adaptation dynamics which arise from the modification of the weights is one source of dynamics in neural-network models.

Processing dynamics which arise from the changes in the data-transfer values due to the decision function is another source of dynamics in neural-network models. Variations in the decision function are due to variations in the activation function and the threshold function. The notation for these functions follows. At time, t, each unit, i, determines its output data-transfer value, $x_i(t)$ based on the threshold function, f, of the unit's activation level, $u_i(t)$, and other variables. The activation level of the unit is, in turn, based on a polynomial activation function, g, of the sum of the output data-transfer values of other units weighted by the values, w_{ji}, and other variables.

$$x_i(t) = f\,(u_i(t), T_i, MAX, MIN) \tag{1.1a}$$

$$u_i(t) = g\,(\sum_i w_{ji} x_j(t-1),\, u_i(t-1),\, A\,,\, B\,,\, C\,,\, D\,,\, \varepsilon) \qquad (1.1b)$$

The terms in the activation function vary between neural-network models. In most models, the sum of the weighted inputs is the primary term. The use of the previous activation level modified by the parameter, A, in some models causes the state of the units to change more smoothly when the inputs are subject to noise. ε is another parameter used in some models for this purpose. Minimum and maximum activation levels determined by the parameters B,C and D, are another method of avoiding inappropriate activation levels. The threshold function transforms the activation level into a range of the data-transfer values for a unit within maximum, MAX, and minimum, MIN. The threshold parameter, T_i is used to adjust the level of activation required to generate a positive data-transfer value from the unit. The threshold function is either a discrete or continuous transformation of the activation level. Continuous threshold functions allow the model to change states more smoothly for activation levels of the units close to the threshold level. Some examples of threshold functions are in figure 1.3.

A simple example of processing dynamics in a three unit neural-network model with lateral connections follows. Let the values of the weights be constant so no adaptation dynamics are involved. The example matrix is:

$$w = \begin{bmatrix} 0 & 1/2 & 1/3 \\ 1/2 & 0 & 1/4 \\ 1/3 & 1/4 & 0 \end{bmatrix}$$

with eigenvalues of 0.733, -0.510 and -0.233, which all have magnitude less than one. Let the activation function for each unit be:

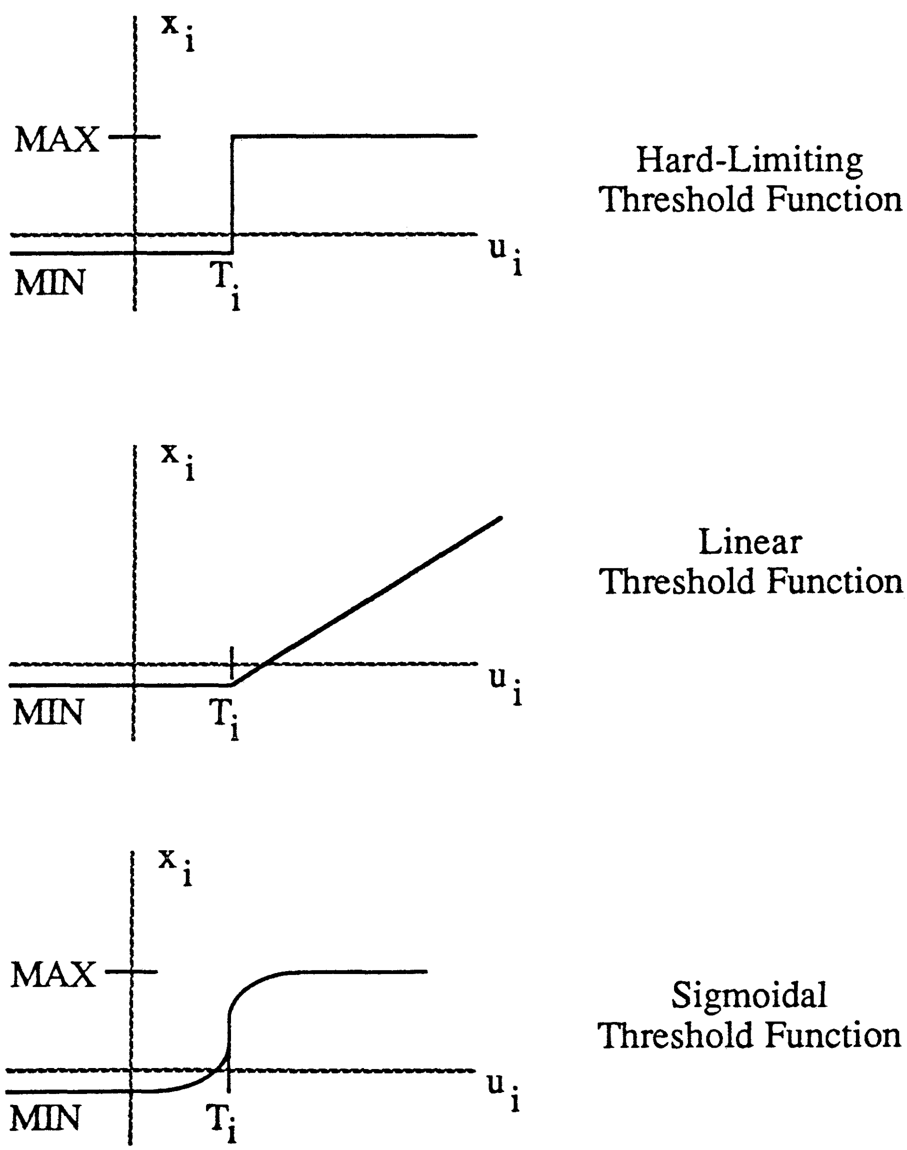


Figure 1.3 Forms of Threshold Functions

$$u_1(t) = \frac{1}{2}\, x_2(t-1) + \frac{1}{3}\, x_3(t-1);$$
$$u_2(t) = \frac{1}{2}\, x_1(t-1) + \frac{1}{4}\, x_3(t-1);$$
$$u_3(t) = \frac{1}{3}\, x_1(t-1) + \frac{1}{4}\, x_2(t-1);$$

with a hard-limiting threshold function with MIN equal to zero and MAX equal to one and T_i equal to two thirds for all the units. The Liapunov function is:

$$E(t) = u_1(t)\, x_1(t) + u_2(t)\, x_2(t) + u_3(t)\, x_3(t)$$

The values over time for the data-transfer values of each unit and the Liapunov function are shown in figure 1.4 with a hard-limiting threshold function and with no threshold function. With synchronous update of the units and no threshold function, the model is an iterative vector matrix multiplier. Since the magnitude of the eigenvalues of w are less than one, then as the number of iterations becomes large, the value of all of the units will be zero.

$$\underline{x}(\infty) = \lim_{t \to \infty} w^t\, \underline{u}(t) = \underline{0}$$

The non-linearity of the threshold function accelerates this process, and allows for stable computation of matrices with eigenvalues with magnitudes greater than one.

The activation function in its simplest form is a vector-matrix multiplier,

$$u = w\, \underline{x}$$

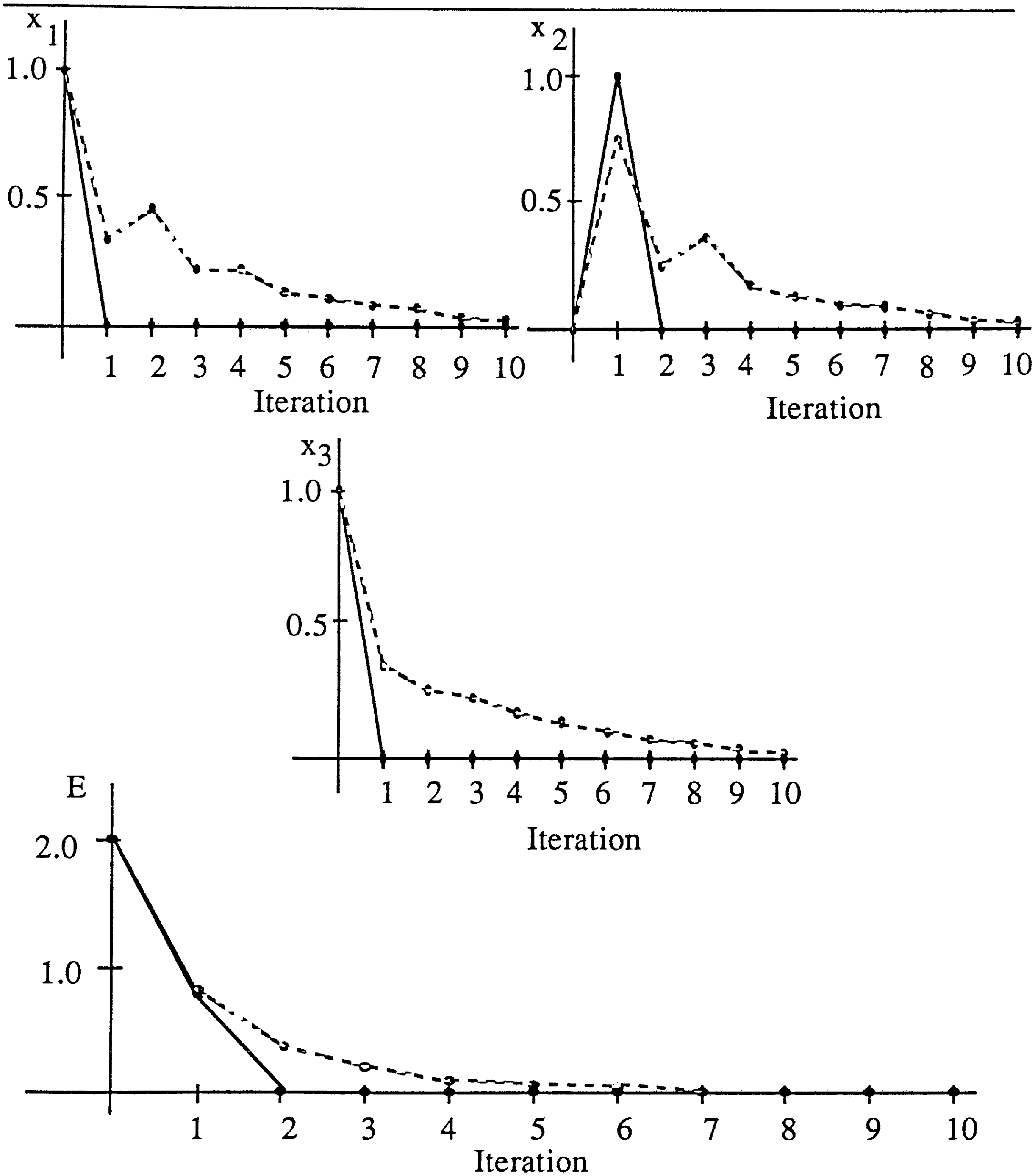


Figure 1.4 Evolution of a Simple Model with Linear and Non-Linear Threshold Functions

but the non-linearity of the transfer function gives neural-network models more processing capabilities than a simple vector-matrix multiplier. An example of a simple feed-forward neural-network model shows the appropriateness of using a vector-matrix multiplier as the activation function. Let the values of the matrix, w, be the sum of the outer product of the external input vectors, $\underline{I}_s$, and the desired output vectors, $\underline{d}_s$.

$$w = \sum_s \underline{d}_s \cdot \underline{I}_s^T$$

This matrix corresponds to the weight matrix in the pattern association application of a neural-network model. If all of the external input vectors are orthogonal and binary, and $\underline{I}$ equals one of them, then the result of the vector-matrix multiplication is:

$$w \cdot I = |\underline{I}|^2 \underline{d}$$

where $\underline{d}$ is the associated desired output vector. The desired output vector is recalled with the neural-network model after the threshold function is used to remove the constant, $|\underline{I}|^2$. The ability of the neural-network model to produce the desired output vector depends upon the vectors being binary since the threshold function essentially only produces two meaningful data-transfer values, MAX and MIN. Thus, by removing the constant, neural-network model improves upon the vector-matrix multiplier in the binary pattern associator application.

The non-linear threshold function applied to the vector-matrix multiplier also provides an advantage when using more than one layer of units. Without the thresholding function, for two layers with feed-forward connection weight matrices *wh* and *wo* and external input vector $\underline{I}$,

$$wo \cdot (wh \cdot \underline{I}) = w \cdot \underline{I}$$

where

$$w = wo \cdot wh$$

No advantage exists for using two layers since both matrices representing the information in the layers can be recoded into one matrix. With the non-linear threshold function, however, the result produced by two layers is different from that produced by one.

$$f(wo \cdot f(wh \cdot \underline{I})) \neq f(w \cdot \underline{I})$$

Additional matrices in the neural-network model which cannot be included in the vector-matrix multiplier alone store additional information about the application. Neural-network models improve upon the highly parallel computation ability of vector-matrix multipliers in performing applications such as pattern association by using non-linear computations and by storing more information.

The salient features of neural-network models are their ability to perform stable, massively parallel, nonlinear computations, and their ability to be "programmed" or adapted using examples of the expected inputs. Neural-network models which performed nonlinear processing and adapted to provide the desired responses were first developed in the early 1960's under the fields of perceptron research[3] and cybernetics[4] . The work in this era attempted to show that stability is obtainable in large systems of simple, independently operating units similar to neurons in the brain. The research performed at this time developed the foundations of modern systems engineering: control theory, automata theory, switching theory and methods of determining stability, such as with Liapunov's second method. Unfortunately, the research on the perceptron met with much less success than these other theories. "Intelligent" operation of the perceptron was difficult to ascertain, and eventually refuted by Marvin Minsky and Seymour Papert in 1969[5] . In 1982, John Hopfield was able to revive general interest in neural-network modeling by showing that even if neural-network models are not "intelligent", many useful applications still exist[6] . Hopfield's model provided a good solution to the computationally difficult traveling salesman problem. The model also performed auto-association for recognizing incomplete input vectors. Hopfield's presentation occurred at the same time as a renewed interest in massively parallel processing in general. Soon afterward, exciting applications of neural-network models began to appear in the literature. These applications included Terrence Sejnowski's presentation of a neural-network model which learned to pronounce text, and Carver Mead's implementation in VLSI of a neural-network model of artificial retina. Table 1.1 shows some other milestones in the development of neural-network models.

Currently, researchers are developing, simulating and implementing more than a dozen different neural-network models without a standard for discussion. The analyses presented here of four neural-network models uses the uniform set of terms for the functional components outlined above, along

Table 1.1 Some Milestones in Neural-Network Modeling

Date	Author	Accomplishment
1949	D.O. Hebb	Postulation of the Hebbian learning rule for the relative effects of neuron input and output on the adaptation of weights.
1958	F. Rosenblatt	The perceptron model as the first neural-network model.
1969	M. Minsky and S. Papert	Refutation of usefulness of perceptrons.
1980	S. Grossberg	Adaptive resonance theory
1982	J.J. Hopfield	First outer-product model and regeneration of general interest in the field.
1984	J.J. Hopfield	Second outer-product model.
1985	D.H. Ackley, G.E. Hinton, T.J. Sejnowski	Boltzman machine
1986	T.J. Sejnowski, C.R. Rosenberg	NETalk
1986	D.E. Rumelhart, G.E. Hinton, R.J. Williams	Backpropagation
1987	C. Mead	VLSI implementation of artificial retina

with the Liapunov functions for unified analyses of the models' dynamics.

1.2 Background on Liapunov Functions

Liapunov functions are widely used in the fields of physics, engineering and mathematics, but generally under different names such as energy or error functions. Control theory is one of the only fields which explicitly uses the term Liapunov function in honor of the mathematician who developed the method of stability analysis using these functions. In A.M. Liapunov's classic book, *Stability of Motion[7]* , two method of stability analysis are presented. The first method determines the stability of a system from the general or particular solution for the equations describing the state of the system's components. The number of interacting components in neural-network models can be in the millions, so the first method is computationally prohibitive in these analyses. The second method determines asymptotic stability of a system of interacting components without explicitly analyzing the dynamics of each component. The number of variables for assessing the stability of a system is reduced from one for the state of each component, to one for the state of the system as a whole, referred to as the value of the Liapunov function. The second method of Liapunov is useful if the individual dynamics are unimportant and a Liapunov function of the overall state of the components can be found. To show that the conditions of the second method are satisfied, the time behavior of the Liapunov function must be known. This restriction implies that all sources of dynamics which cause changes in the Liapunov function are known. If the sources of the dynamics are overcome by dissipative forces in the system, then the system is asymptotically stable.

Asymptotic stability is important in neural-network models if the models are to solve problems in a finite amount of time. The neural-network models used in neurobiological research which ignore this requirement of stability fall outside the scope of this method. Liapunov's second method is applicable to neural-network models since the units in a neural-network model form a system of interacting components which would be difficult to analyze discretely. The loss of information about the individual dynamics of the units is unimportant since the evolution of the states of the individual units over time is much less important than the final state of the model. The time behavior of the Liapunov functions for the neural-network models analyzed in this thesis is known from previous research in other fields. Therefore, the use of Liapunov's second method to analyze the stability of neural-network models is appropriate.

The derivation of Liapunov's second method is fairly complex, but the results are easy to state, assuming that a Liapunov function of the states of the systems is known. Formally, Liapunov's second method gives three conditions for a Liapunov function, E, of a stable dynamic system with state, $\underline{x}(t)$, at time t and state $\underline{x}_{eq}$ at equilibrium. $\underline{x}(t)$ is a vector of the states of the system components. The conditions are:

(1) $E(\underline{x}, t)$ *is differentiable with respect to time.* (1.2a)

(2) $E(\underline{x}, t) > 0$ for $\underline{x} \neq \underline{x}_{eq}$ and $E(\underline{x}_{eq}) = 0$. (1.2b)

(3) $\frac{\partial}{\partial t} E(\underline{x}, t) < 0$ for $\underline{x} \neq \underline{x}_{eq}$. (1.2c)

If these conditions are satisfied, then the system is asymptotically stable. However, violation of the conditions does not necessarily imply that the system is unstable. If no Liapunov function is found which satisfies the conditions, then another method of stability analysis is used, such as direct solution of the dynamical equations which describe the system, or simulation of the system under the expected conditions. When a Liapunov function is known, the second method is certainly the easiest approach to stability analysis. An example of the form of a Liapunov function is shown in figure 1.5.

Not all Liapunov functions strictly adhere to the above conditions. Equivalent representations of a Liapunov function are for $E(\underline{x},t)$ to be negative and its derivative positive, or for $E(\underline{x},t)$ to be bounded and monotonically increasing or decreasing. Figures 1.6 and 1.7 give examples of the form of these equivalent Liapunov functions. Simple transformation of the alternate Liapunov functions yield the original conditions. Negation of the negative Liapunov function is all that is required to meet the original conditions, but in thermodynamics, the convention is for E to be negative. Transformation of the original conditions of Liapunov's second method to conform to this convention yields:

(1) $E(\underline{x},t)$ *is differentiable with respect to time.* (1.3a)

(2) $E(\underline{x},t) < 0$ for $\underline{x} \neq \underline{x}_{eq}$ and $E(\underline{x}_{eq}) = 0$. (1.3b)

(3) $\frac{\partial}{\partial t}E(\underline{x},t) > 0$ for $\underline{x} \neq \underline{x}_{eq}$. (1.3c)

Likewise, the convention in the study of the spin-glass model of the dynamics of a system of disordered magnetic material is for E to be bounded.

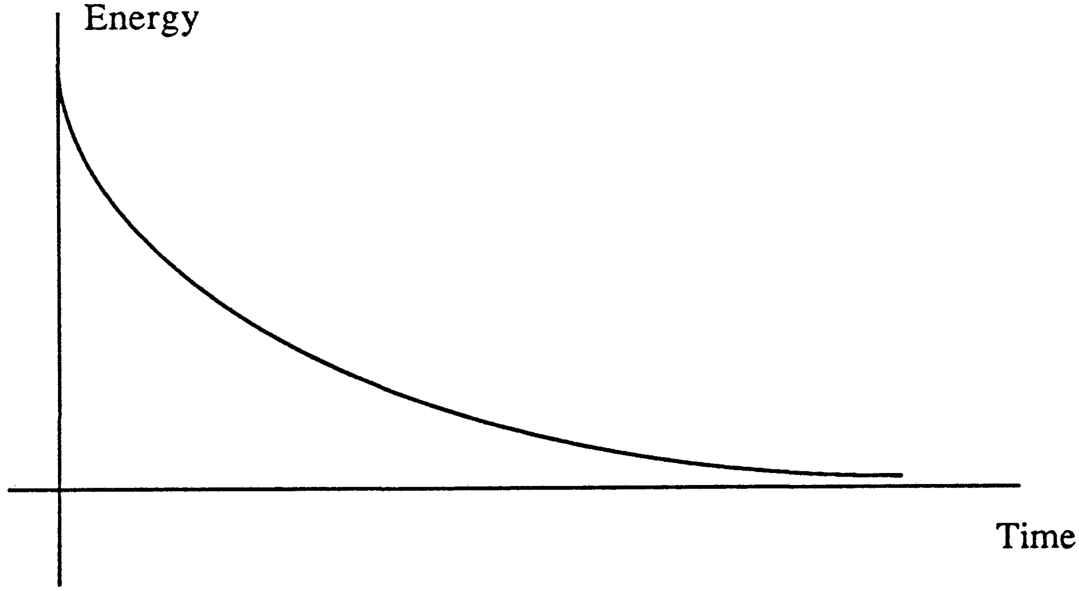


Figure 1.5 Example of an Asymptotically Stable Liapunov Function

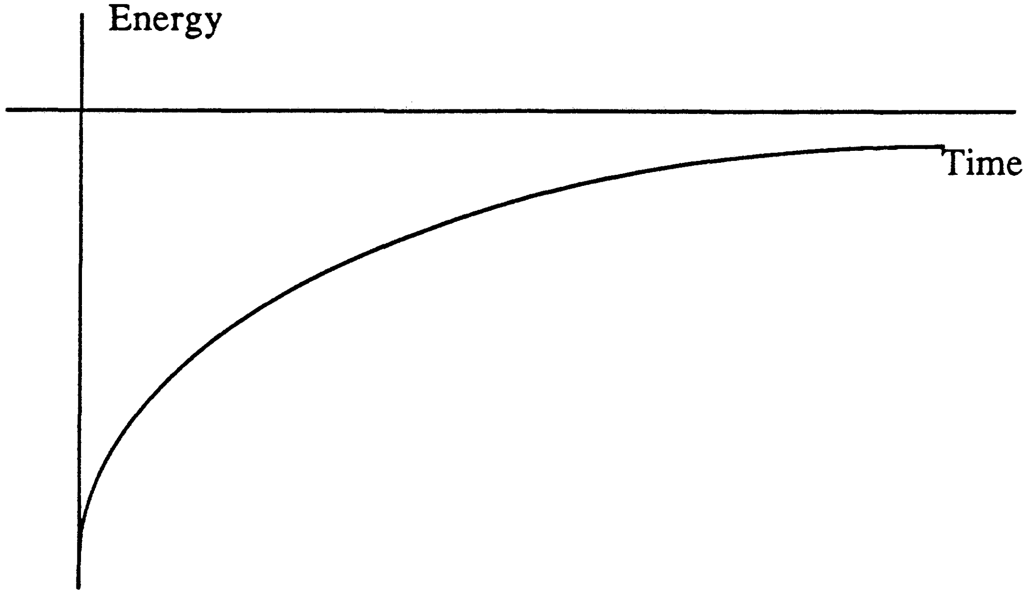


Figure 1.6 Example of a Negative Liapunov Function

Subtraction of the maximum for monotonically increasing functions, or subtraction of the minimum for monotonically decreasing functions reduces E to the form of Liapunov's second method. The conditions for Liapunov's second method under this transformation are:

(1) $E(\underline{x}, t)$ *is differentiable with respect to time.* (1.4a)

(2) $E(\underline{x}, t)$ *is bounded* for $\underline{x} \neq \underline{x}_{eq}$

and $E(\underline{x}_{eq})$ *equal to one of the bounds.* (1.4b)

(3) $E(\underline{x}, t)$ *is monotonically increasing or decreasing*

for $\underline{x} \neq \underline{x}_{eq}$. (1.4c)

The choice of representation of a Liapunov function does not affect the analysis of neural-network models, since all three representations are

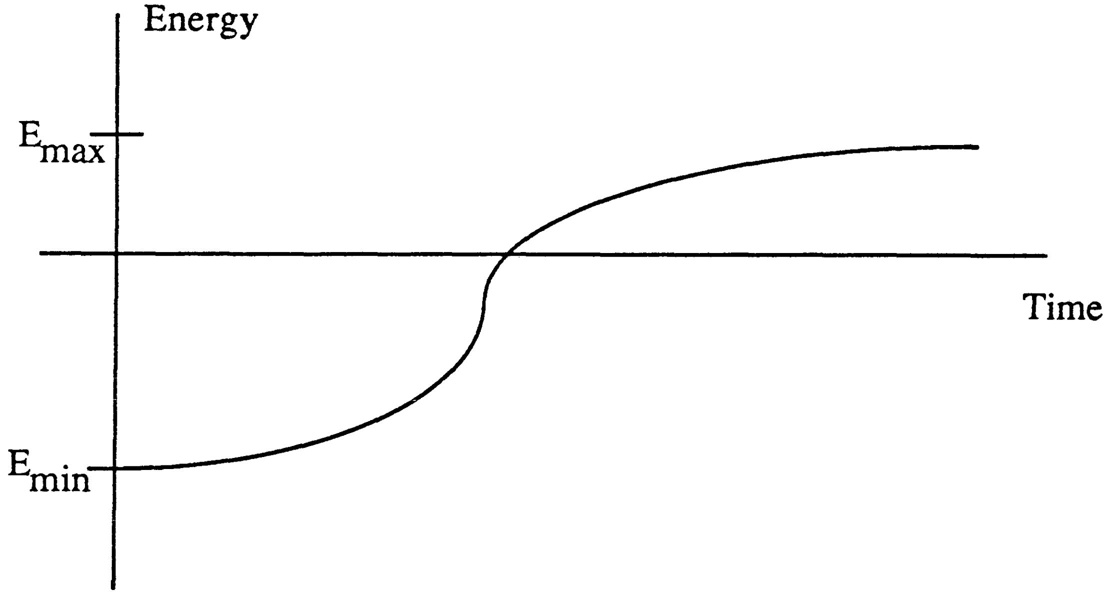


Figure 1.7 Example of a Bounded Liapunov Function

functionally equivalent. The choice of form is mostly dependent upon the model's originating field of research.

Discrete Liapunov functions are another variation on the original conditions set forth by Liapunov with the derivative in the last condition replaced by the discrete change in E. The first condition is changed to the condition that the Liapunov function is a reasonable approximation of a continuous function. The degree to which this condition is met is dependent upon the specific application and period of operation for the system. For the discrete standard Liapunov function, the conditions are:

(1) $E(\underline{x}, t)$ *is a reasonable approximation to a continuous function.* (1.5a)

(2) $E(\underline{x}, t) > 0$ for $\underline{x} \neq \underline{x}_{eq}$ and $E(\underline{x}_{eq}) = 0$. (1.5b)

(3) $\Delta E(\underline{x}, t) < 0$ for $\underline{x} \neq \underline{x}_{eq}$. (1.5c)

Likewise, for the discrete negative Liapunov function, the conditions are:

(1) $E(\underline{x}, t)$ *is a reasonable approximation to a continuous function.* (1.6a)

(2) $E(\underline{x}, t) > 0$ for $\underline{x} \neq \underline{x}_{eq}$ and $E(\underline{x}_{eq}) = 0$. (1.6b)

(3) $\Delta E(\underline{x}, t) > 0$ for $\underline{x} \neq \underline{x}_{eq}$. (1.6c)

and for the discrete bounded Liapunov function, the conditions are:

(1) $E(\underline{x}, t)$ *is a reasonable approximation to a continuous function.* (1.7a)

(2) $E(\underline{x}, t) > 0$ for $\underline{x} \neq \underline{x}_{eq}$ and $E(\underline{x}_{eq}) = 0$. (1.7b)

(3) $E(\underline{x}, t)$ *is monotone increasing or decreasing* for $\underline{x} \neq \underline{x}_{eq}$. (1.7c)

Discrete neural-network models are approximations of a continuously operating system of neurons. The assumption is made here that these approximations are appropriate and the change in the Liapunov function, ΔE, is small enough to be meaningful for the model. The analyses in the chapters which follow use discrete Liapunov functions wherever the model is specified as operating discretely. Some modifications to the functions derived from the Liapunov function are made to improve the approximation to continuous processing.

To date, no simple basis for choosing a Liapunov function is available other than experience with similar systems and a sense of the behavior of the system. For several of the neural-network models analyzed here, an existing physical model or algorithm was used as the basis for the Liapunov function. For these models, the dynamics satisfy the conditions of Liapunov's second method in the same manner as the original system. The outer-product model uses a Liapunov function derived in physics for spin-glass problems. Backpropagation uses a Liapunov function derived in signal processing for least-mean squares algorithms. The Boltzman machine uses two Liapunov functions, one from thermodynamics and one from information theory. The Liapunov functions used here for adaptive resonance theory was derived independently by this author using methods employed in control theory. The originators of adaptive resonance theory are reputed to have developed Liapunov functions[8] , but the functions are not obviously available and the derivations are not difficult. The Liapunov functions for each of these neural-network models have different forms, but the functions play a similar role of defining the constraints for the functional components in each model.

The influence of Liapunov functions in the neural-network models is masked by the terms and representations used. The names and the representations from the originating fields for neural-network models are different for each of the Liapunov functions. Energy is the name used both for the negative Liapunov function in thermodynamics and for the bounded Liapunov function in the study of spin-glass models. Information gain from information theory and mean-squared error from optimization theory are names used for standard Liapunov functions. The components in the originating fields which are analogous to the vector of the data-transfer values of the units, $\underline{x}$, in the neural-network models are momentum in thermodynamics, polarization in spin-glass models, probability in information theory and distance from an optimal condition in optimization theory.

The Liapunov functions provided or suggested by the authors of the representative neural-network models are the starting point for the analyses in Chapters 2-7. This thesis shows the importance of Liapunov functions in neural-network models, and points out assumptions and parameter restrictions which are necessary to maintain the conditions of Liapunov's second method.

Chapter 2
ANALYSIS OF OUTER-PRODUCT MODEL

2.1 Overview of Outer-Product Model

The analysis of models presented in this thesis begins with an analysis of the model with the simplest dynamics, the outer-product (Hopfield) model. Only processing dynamics derived from the spin-glass Liapunov function are used since the values of the weights are pre-determined by the application. The source of the dynamics for this neural-network model is displacements of the model from equilibrium caused by changes in the data-transfer values of the units. The operation of the model is instigated by the user causing the data-transfer values to differ from their current equilibrium values. The user obtains a solution by observing the data-transfer values once a new equilibrium is found by the model. This type of processing is useful for pattern completion (also referred to as auto-associative processing) and optimization. The objective of this analysis is to use Liapunov functions to show under what conditions a stable equilibrium is reached.

Both the discrete and the continuous versions of the outer-product model are analyzed here to show how the components and restrictions on parameters differ. The interconnections between units, weights, and decision function are simple in both versions. The only parameters in the models are the constant weights and the threshold level in the discrete model, plus the activation function parameters in the continuous model. The connection component is identical in both versions.

The outer-product model can be connected in either one or many layers of units. Each unit is connected laterally to every other unit in its layer, and/or each unit has feed-forward and feedback connections to each unit in the the next layer and the preceding layer of multilayer models. For analysis purposes, the model is assumed to have a single, laterally connected, layer with each unit an output unit (see figure 2.1). The use of the external input vector in the model is vague. A vector, referred to here as $\underline{I}'$, is used in the literature as an "input bias," but this parameter is frequently ignored in analysis, even by Hopfield. The external input vector is instead the initial state of the data-transfer values, $x_i(0)$. for external input vector $\underline{I_s}$ of the set of external input vectors for the application,

$$x_i(0) = I_{si} \text{ for } all\ units, i$$

This assumption about the use of the external input vector is adopted in this analysis.

The outer-product model has two primary types of applications: pattern completion and optimization. For both types of applications, the values of the weights are symmetrical, i.e. $w_{ji} = w_{ij}$ and no direct feedback is provided to the unit, i.e. $w_{ii} = 0$.

Pattern completion applications process partial or corrupted vectors to generate the corresponding known vector. For an outer-product model implementation of this type of application, the values of the weights are the sum of the outer-products of all of the external input vectors to be recognized by the model. The name outer-product model arises from the use of

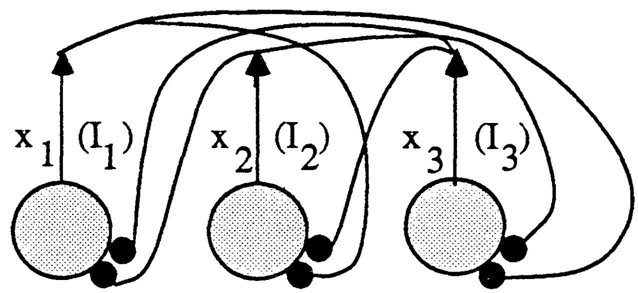


Figure 2.1 Functional Components of Outer-Product Model

this type of weight determination. The vectors recorded, $\underline{I}_s$, need to be binary and orthogonal. This need for this constraint is illustrated mathematically:

$$w = \sum_s \underline{I}_s \cdot \underline{I}_s^T$$

where w is the weight matrix. The processing in the outer-product model approximates a series of presentations of an input vector $\underline{I}$:

$$\underline{u} = w \cdot \underline{I} = \sum_s \underline{I}_s \cdot \underline{I}_s^T \cdot \underline{I}$$

If $\underline{I}$ is one of the orthogonal vectors which are recorded in the weights, then

$$\underline{u} = |\underline{I}|^2 \underline{I}$$

which is a constant times the learned vector. The threshold function removes the effect of the constant for binary input vectors. For instance, if $\underline{I}$ is a vector with values of the elements (-1,+1), then a hard-limiting function with MIN = -1, MAX = +1 and $T_i = 0$ would cause $\underline{x}$ to equal $\underline{I}$ when $|\underline{I}|^2$ is any positive constant. A similar sigmoid threshold function which rapidly approaches these bounds would work well also. For $\underline{I}$ a vector with values of the elements (0,+1), the threshold function is more difficult to optimize. For a hard-limiting threshold function with MIN = 0 and MAX = +1 the threshold level is restricted to:

$$0 < T_i < \min_s |\underline{I}_s|^2$$

The restrictions on the threshold level for a similar sigmoid threshold function are

$$T_i \approx \frac{\min_s |\underline{I}_s|^2}{2}$$

Note that the vector-matrix multiply followed by the threshold function is performed more than once for a single external input vector as the model settles to an equilibrium point. The iterations which occur in the outer-product model damp out slight variations in the vector being recognized which allows the external input vector to be slightly corrupted or "noisy". The extent to which a vector can deviate from the recorded orthogonal vectors is still an open question.

For optimization applications, the weights do not need to be based on orthogonal vectors. Optimization applications have a set of constraints, or costs, for elements of the system. Processing for this type of application solves for the state of the elements of the system which results in the lowest total cost. The values of the weights are the cost, or penalty, for adjacent units having positive data-transfer values. The costs are reciprocal so the weights are symmetrical with w_{ii} equal to zero. An example of an optimization application which is successfully solved using the outer-product model is the traveling salesman problem. The solution of the traveling salesman problem consists of finding the shortest route between a set of cities. The weights are set equal to the distance between the cities[9] . If the distance is large, then the cities should not follow each other in a route; i.e. two units representing the order of the cities should not both have large data-transfer values when the the weight connecting them is large. The model uses the relative sizes of the weights to find a route which has a short total distance (see figure 2.2).

The decision function ensures that the model's dynamics are dissipative and the conditions for Liapunov's second method are satisfied. The decision function is different in the discretely and continuously operating versions of the outer-product model. The discrete version has binary data-transfer values for each unit generated asynchronously by a a hard-limiting threshold function of the current activation level. The continuous version has continuous, bounded data-transfer values for each unit generated synchronously by a sigmoidal threshold function of current and proceeding activation levels. Both the discrete and the continuous versions of the outer-product model will be analyzed in sections 2.2 and 2.3 to show the differences in the resulting Liapunov functions and restrictions on the parameters of the models.

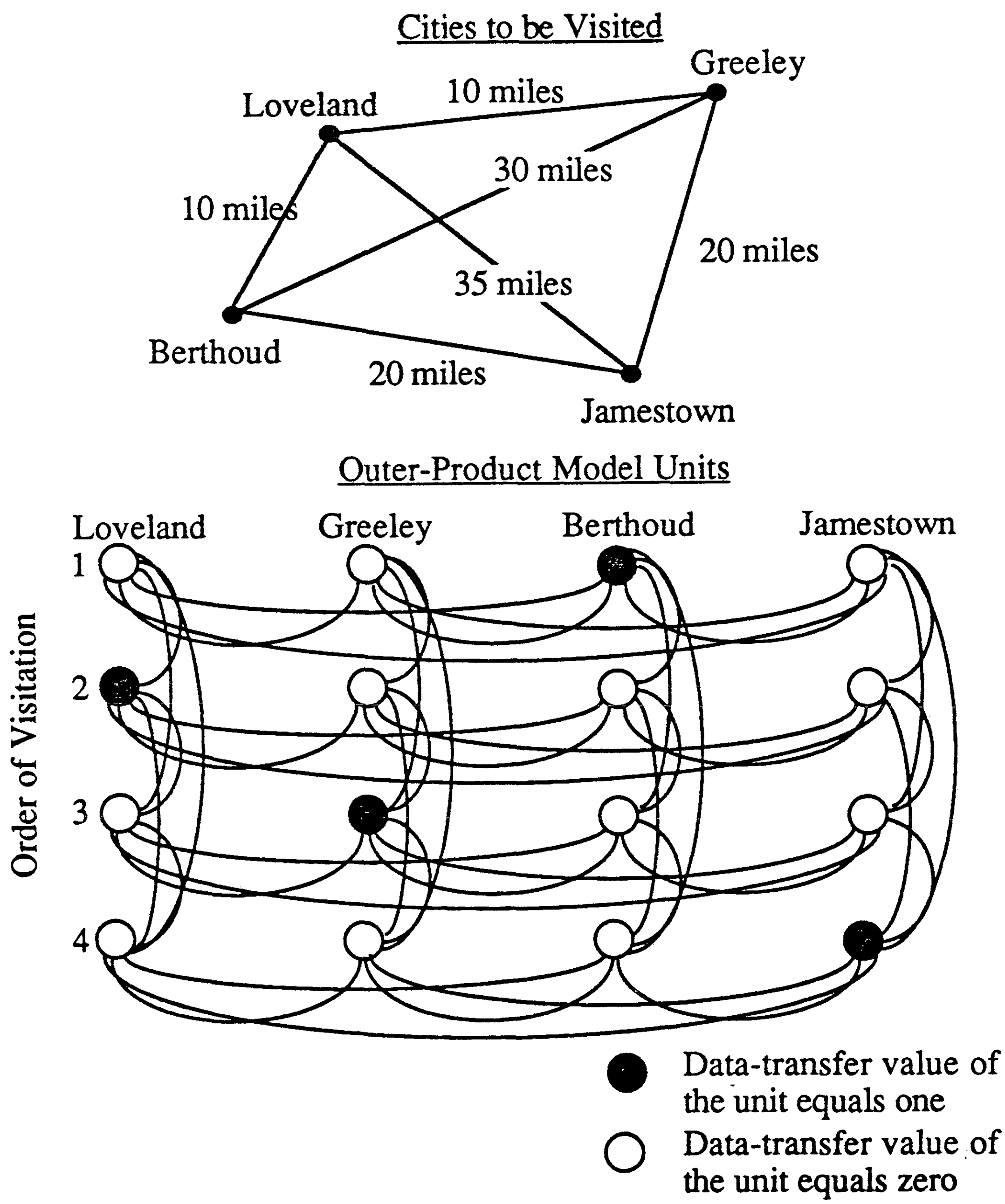


Figure 2.2 Outer-Product Model for Solution of the Traveling Salesman Problem

2.2 Analysis of Stability of the Discrete Outer-Product Model

The analysis of the stability of the discrete outer-product model is the starting point for this analysis of the restrictions on parameters caused by the stability requirement. The stability of the discrete outer-product model is demonstrated in some detail in Hopfield's 1982 paper, but the implied restrictions on parameters are not pointed out as in this analysis. A review of Hopfield's stability analysis is presented here before a more in depth analysis is performed for the model in general and for common types of applications.

The Liapunov function for the outer-product model arises from the spin-glass model. The functioning of the units in the outer-product model is very similar to change in the states of the elements in the spin-glass model at zero temperature[10] . The first outer-product model is binary, as is the spin-glass model. Each unit can have one of two output states with values [MIN, MAX] for data-transfer. The weights in the outer-product model are analogous to the interaction strengths between adjacent dipoles in the spin-glass model. The values of the weights are constants determined by the application before the model is used for processing. One difference in the outer-product model is that the change in the output state is a function of the output data-transfer values of all of the other units in the network, instead of just the neighboring units in the spin-glass model.

Using the spin-glass Liapunov equation as a guideline, the following discrete Liapunov equation is applied to the discrete outer-product model.

$$E(t) = -\frac{1}{2}\sum_{i \neq j}\sum w_{ij}\, x_j(t)\, x_i(t) - \sum_i I'_i\, x_i(t) + \sum_i T_i\, x_i(t)$$

Since the input bias, I'_i and the threshold level, T_i are both parameters set before the operation of the model begins, T'_i can be defined such that

$$T'_i = I'_i - T_i$$

is the effective threshold level. The Liapunov function can then be rewritten as

$$E(t) = -\frac{1}{2}\sum_{j\,!=i}\sum w_{ji}\, x_j(t)\, x_i(t) - \sum_i T'_i\, x_i(t)$$

without loss of generality. The data-transfer values, x_i are bounded and all other parameters are constant, so this is a bounded Liapunov function. The discrete outer-product model is stable since conditions 1.7 of Liapunov's second method are shown here to be satisfied.

The units are updated asynchronously, so at most one output data-transfer value will change at a given time. The change in Liapunov function when one output data-transfer value is changed, assuming the values of the weights are symmetrical, is:

$$E(t) - E(t-1) = \Delta E(t) = -\sum_i \Delta x_i(t)\, [\, \sum_{j \neq i} w_{ji}\, x_j(t-1) + T'_i\,]$$

where

$$\Delta x_i(t) = x_i(t) - x_{(t-1)}$$

The threshold function which produces the data-transfer value, x_i for each unit i at time, t, is:

$$x_i(t) = MAX \text{ if } u_i(t) > 0$$

$$x_i(t) = MIN \text{ if } u_i(t) < 0$$

where MAX and MIN are the values for the hard-limiting threshold function and the activation level, $u_i(t)$ is

$$u_i(t) = \sum_{j \neq i} w_{ji}\, x_j(t-1) + T'_i$$

The change in Liapunov function can therefore be rewritten as:

$$\Delta E(t) = -\sum_i \Delta x_i(t)\, u_i(t)$$

The decision function was chosen so that the change in the Liapunov function is always non-positive to satisfy condition 1.7c. When $u_i(t)$ is negative, and $x_i(t-1)$ has value MAX, $\Delta x_i(t)$ is negative and product of $u_i(t)$ and $\Delta x_i(t)$ is positive. When $u_i(t)$ is negative and $x_i(t-1)$ has value MIN, $\Delta x_i(t)$ is zero and the product is zero. Likewise, when $u_i(t)$ is positive, and $x_i(t-1)$ has value MAX, $\Delta x_i(t)$ is zero and product of $u_i(t)$ and $\Delta x_i(t)$ is zero. When $u_i(t)$ is positive and $x_i(t-1)$ has value MIN, $\Delta x_i(t)$ is positive and the product is positive. In any of these cases, $\Delta x_i(t)\, u_i(t)$ is non-negative, so $\Delta E(t)$ is non-positive and condition 1.7c is satisfied. Conditions 1.7a and 1.7b are satisfied since the Liapunov function is bounded and the change in the Liapunov function is small when the number of units is large. Therefore, the outer-product model is asymptotically stable.

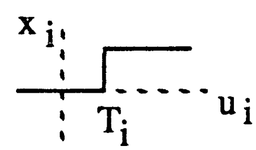
The analysis is now carried beyond that performed by Hopfield in order to show the restrictions on the parameters implied by the Liapunov function (see table 2.1). The choice of the spin-glass Liapunov function defines the data-transfer values to be binary, the weights are defined to be constant and symmetrical with w_{ii} equal to zero, and the decision function is defined to be hard- limiting. Multiple layers of units are permissible, as long as the weights are symmetrical. The analysis does not directly specify the values for the weights, I'_i, or T_i as long as these values are constant. A large number of units are required for the the discrete approximation to be valid. The exact number of units depends on the application and the characteristics of the specific Liapunov function.

The restrictions on the parameters arising from this analysis are not necessary, but the restrictions are sufficient for implementing a model without conducting an extensive parameter analysis. The Liapunov functions used in the discrete and continuous versions of the outer-product model are similar enough that similar restrictions on parameters arise from the analysis of the continuous outer-product model presented in the next section.

2.3 Analysis of Stability of the Continuous Outer-Product Model

The analysis for the continuous outer-product model is performed in a manner identical to the analysis for the discrete model. Hopfield's 1984 stability analysis is included to show how the model and the Liapunov function were modified from the discrete case. The analysis is then expanded to show the restrictions on the parameters in general and for some common types of application. The analysis of these restrictions yields values for parameters in an continuous outer-product model which aid in stablizing the processing of the model. A similar approach was taken by A.N. Michel, et

Table 2.1 Restrictions on Discrete Outer-Product Model

Connections	Weights and Adaptation	Decision Function
1) Multiple or single layers of units	1) No adaptation - Constant valued weights	1) I'_i constant
2) Large number of units	2) Symetrical weights ($w_{ij} = w_{ji}$)	2) T_i constant
	3) No direct feed-back ($w_{ii} = 0$)	3) Asynchronous change in data transfer values
		4) Hard-limiting threshold function 
		5) Activation level is proportional to weighted outer-product of data transfer values
		6) No memory in activation level

al. to estimate the stable states of a continuous outer-product model[11] .

The argument for stability directly based on the discrete spin-glass Liapunov function does not hold for the continuous model. To show stability for the continuous model, changes must be made to both the Liapunov function and the decision function. The resulting Liapunov function is still very similar in form to the spin-glass Liapunov function and bounded with the upper and lower bounds on the data-transfer values. The Liapunov function for the continuous outer-product model is:

$$E(t) = -\frac{1}{2} \sum\sum_{i \neq j} w_{ij}\, x_i(t)\, x_j(t) + \sum_i A_i \int_0^{x_i(t)} f^{-1}(x)\, dx - \sum_i I'_i\, x_i(t)$$

where A_i is a constant from the decision function. The derivative of this Liapunov function is shown to be non-positive which satisfies the stability criterion 1.4c for monotonicity of the derivative of the Liapunov function. This continuously non-positive derivative is obtained by modifying the decision function. The time derivative of the Liapunov function is:

$$\frac{dE}{dt} = -\sum\sum_{i \neq j} w_{ij}\, x_j(t)\, \frac{dx_i}{dt} + \sum_i A_i\, f^{-1}(x_i(t))\, \frac{dx_i}{dt} - \sum_i I'_i\, \frac{dx_i}{dt}$$

To force the derivative to be negative, the decision function is modified. The activation function for the is:

$$\varepsilon_i\, \frac{du_i}{dt} = \sum_j w_{ij}\, x_j(t) - A_i\, u_i(t) + I'_i$$

where ε_i and $1/A_i$ are constants analogous to capacitance and resistance in electrical circuits. The output data-transfer value is determined by thresholding the activation level,

$$x_i = f_i(u_i)$$

where $f_i(x)$ is a continuous monotone increasing function. The time derivative can then be simplified to:

$$\frac{dE}{dt} = -\sum_i \varepsilon_i \frac{du_i}{dt} \frac{dx_i}{dt}$$

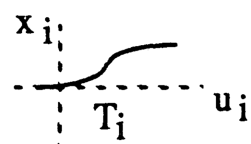
$$= -\sum_i \varepsilon_i f_i'(u_i) \left(\frac{du_i}{dt}\right)^2$$

The time derivative of the Liapunov function is always non-positive since ε_i and $f_i'(u_i)$ are defined to be positive. The system is at a stable point when $\frac{dx_i}{dt}$ =0, or the value of the Liapunov function is at its most negative bound. Thus, Hopfield has shown that the system is stable by criterion 1.4 that bounded Liapunov functions are monotonically non-increasing or non-decreasing.

Further analysis by this author illuminates restrictions on the parameters (see table 2.2). The modified Liapunov function for the second model defines the data-transfer values to be bounded above and below, the weights are defined to be constant, and symmetrical with w_{ii} equal to zero for all i as in the first outer-product model, and ε_i and f'_i are positive. The unrestricted features are similar to the first model; the model may have multiple layers, and the weights between units and the resistance, $1/A_i$, are constants.

The results of this analysis are very similar to the results of the analysis of the discrete model for the restrictions on the parameters. The additional

Table 2.2 Restrictions on Continuous Outer-Product Model

Connections	Weights and Adaptation	Decision Function
1) Multiple or single layers of units	1) No adaptation - Constant valued weights	1) T_i constant
	2) Symetrical weights ($w_{ij} = w_{ji}$)	2) A_i constant
	3) No direct feed-back ($w_{ii} = 0$)	3) $\varepsilon_i > 0$, constant
		4) $f'_i(u_i) > 0$ and $f_i(u_i)$ is bounded 
		5) Synchronous change in data-transfer values
		6) Activation level is proportional to weighted outer-product of data transfer values
		7) Memory in activation level

parameters, ε_i and A_i only had the restriction that they were positive. Since processing is continuous, no restrictions are required on the number of units.

2.4 Summary of Stability Analyses for Outer-Product Model

The main difference in the analysis of the discrete version and this analysis of the continuous version is in the decision function. The discrete version has a threshold function which changes abruptly between the two bounds, while the continuous version changes smoothly between the bounds. The addition of the memory term of $-A_i u_i$ in the activation function of continuous version is not necessary for the development of a model which functioned continuously. The memory term has the advantage of giving the changes in activation level of a unit momentum to overcome short-term fluctuations due to noise in the external input vector or irregularities in the Liapunov function. A similar term could be added to the discrete version for similar benefits. In addition, a memory term would improve the approximation of the discrete model to a continuously operating system by forcing the changes in the activation level to occur more smoothly. The need for a memory term can be determined through further analysis for a specific application. Simulation analysis, similar to that performed in Chapter 6 for the backpropagation model, is one approach for expanding the analysis beyond what was performed here.

The applications for either version of the outer-product model are limited because no provision exists for adapting the weights as the application is processed. The adaptation scheme used in the Boltzman machine is an approach to adaptation in a model similar to the outer-product model. The adaptation dynamics impose additional parameters and parameter restrictions on the model. These are analyzed using a second Liapunov function for the adaptation dynamics which is not affected by the original Liapunov function for processing dynamics. The analysis of these dynamics follows in Chapter 3.

Chapter 3
ANALYSIS OF BOLTZMAN MACHINE

3.1 Overview of the Boltzman Machine

The second model presented for analysis is the Boltzman machine. The Boltzman machine adds adaptation dynamics to a modified version of the discrete outer-product model's processing dynamics. The processing dynamics in the Boltzman machine are similar to the processing dynamics in the discrete outer-product model in that both dynamics have a Liapunov function derived for the spin-glass model. The difference is that the discrete outer-product model removes the stochastic aspects of the spin-glass model by assuming that the temperature which governs randomness is zero. The Boltzman machine takes advantage of the stochastic processing to overcome some of the outer-product model's problems and to suggest a stochastic approach to the weight adaptation. The use of stochastic processing overcomes many of the outer-product model's problems with local minima in the Liapunov function which do not provide information for the application. These minima result from the weights storing information on more than one vector. Some local minima are desirable, such as the local minima which store the information for pattern completion in the outer-product model. In other applications, such as optimization applications, the global minimum holds the information which is sought. Hetero-associative processing in which the input vector is to be associated with a different output vector, is an application in which shallow local minima need to be ignored, but deep local minima are "good" solutions. The stochastic processing of the Boltzman machine causes the Liapunov function to occasionally increase so that it leaves local minima and continues towards stability at the global minimum or a "deep" local minimum. The adaptation harnesses the statistical nature of stochastic processes to determine the weights. By using the Boltzman distribution (from which the name Boltzman machine arises) the model records and recalls patterns using their probability distribution instead of directly computing the sum of the outer products of the vectors to be processed as in the outer-product model. This approach permits the processing of hetero-associative applications in which an external input vector is associated with a different desired output vector.

The Boltzman machine differs from the discrete outer-product model in the connections between units even though the mathematical expressions appear similar. The units in the outer-product model are usually connected laterally while the Boltzman machine model is arranged in multiple layers with feed-forward and feedback connections between layers and no lateral connections within the layers (see figure 3.1). The first layer is the input layer. During processing, the initial state of the units in the input layer is the external input vector. The last layer is the output layer. During processing, the state of the output units is the output vector to the user. The layers between the input and output layers are the hidden layers. The state of the units in these layers are not available to the user, but the use of hidden layers enhances the ability of the model to represent information. Hidden layers enable the model to learn associations such as in the exclusive-or operation.

The connections appear complicated, but mathematically, the number of weight matrices which represent the connections is only one more than the number of hidden layers. This simplification occurs because the feed-forward and feedback connections between a pair of units have identical weights. If only one hidden layer is used then only two weight matrices are needed. The optimal number of hidden layers is an open question.

Weight adaptation in multi-layer neural-network models was one of the most difficult problems for early neural-network researchers. A criterion for changing the weights for units in the hidden layer is difficult to find. The Boltzman machine and the backpropagation model analyzed in Chapter 6 provide two different approaches to this problem. For the Boltzman machine, the criterion is the statistics on the probability that both units connected by a weight have positive data-transfer values. The adaptation for the Boltzman machine is performed in two steps. In the first step the data transfer values of the input units are set equal to the external input vector and the output units are set equal to the desired output vector. Statistics are then gathered on the frequency that the units which a weight connects both have positive data-transfer values. In the second step the data-transfer values for the input and output units are allowed to vary and more statistics are gathered for the weights. Using the difference in statistics computed in these two steps, the weights are adapted for all of the units. This solution for the weights will improve with the number of presentations of external input / desired output vector pairs. The external input vectors are occasionally corrupted with noise to prevent weights from becoming too large in an attempt to represent patterns which have probability zero of occurring. Since

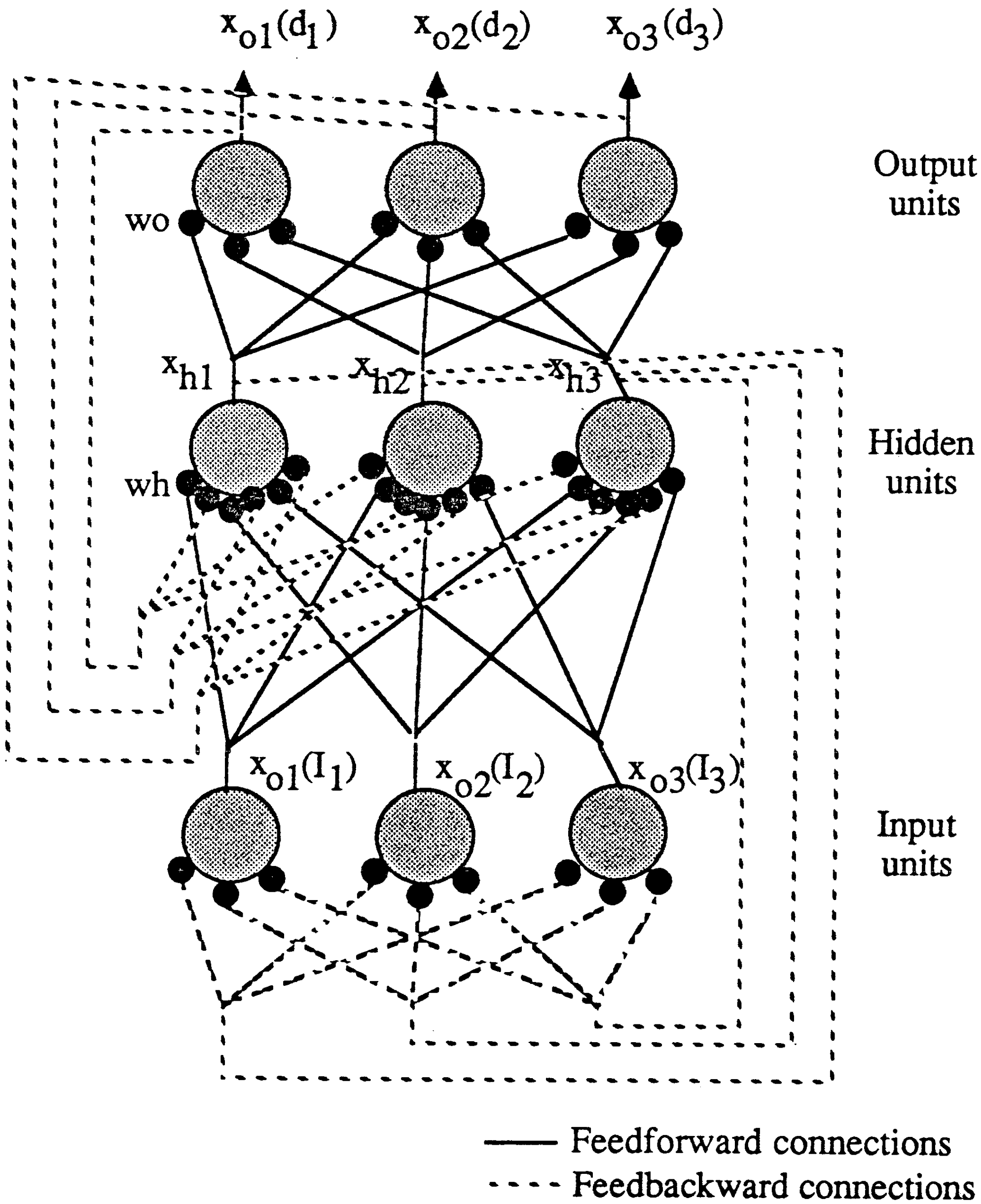


Figure 3.1 Functional Components of Boltzman Machine

statistics are being generated during adaptation, the weight changes will be most stable if adaptation is repeatedly performed for only a limited number of external input / desired output vector pairs. Therefore, most of the processing is performed after the weights are adapted.

The processing of the data-transfer values is performed by a process referred to as simulated annealing. Simulated annealing has shown promise in signal processing applications[12] . The decision function determines the output data-transfer value of the units probabilistically based on the sum of the weighted inputs and a parameter called temperature, T. Temperature is slowly reduced until a stable pattern of output values is obtained. When the temperature is large, the data-transfer values appear to change randomly. The closer T comes to zero, the more the Boltzman machine processes like the discrete outer-product model, which corresponds to the Boltzman machine at zero temperature. Because of this correspondence, the Liapunov function for the processing dynamics and the asynchronous update of the output data-transfer values of the units are the same as in the discrete outer-product model with MIN = 0, MAX = 1, and T'_i set to zero.

The Boltzman machine improves upon the outer-product model at the expense of the large number of iterations required to process information. A large number of iterations are also required to obtain good statistics for adaptation. Speed of operation prevents the Boltzman machine from currently being practical, but as the use of parallel processors increases, this model may become more feasible. The analysis of section 3.2 shows the restrictions on parameters required to make all of these iterations stable.

3.2 Analysis of Stability of the Boltzman Machine

With respect to the stability of the model, the Boltzman machine can be seen as having two very different phases. The first phase has a constant set of data-transfer values and dynamic weight values. The second phase the model has constant weights and dynamic data-transfer values. Since the weights are generally adapted before the processing begins, these two phases are analyzed separately here. The stability of the Boltzman machine is analyzed in two steps: the stability of the system for a learned set of weights and the stability of the weight adaptation. The first step uses the same Liapunov function as the outer-product model. The second step minimizes a function from information theory. The stability analysis for the processing dynamics is very straight forward, but the analysis for the adaptation dynamics requires much more mathematical and statistical manipulation than was required for the outer-product model. Before these analyses are presented, some history of the model is presented.

The Liapunov function of the outer-product model can be seen as a function minimized by a gradient descent method of optimization theory. The next data-transfer value of a unit is the value which maximizes the change towards the global minimum. The primary problem with gradient descent methods is that the model can get caught in local minima in the

Liapunov function which correspond to stable, but sub-optimal solutions. In terms of the conditions of Liapunov's second method, a local minima in the bounded Liapunov function used in the outer-product model occurs when the derivative of the Liapunov function is zero at a point where the function is greater than the lower bound. To overcome these local minima in the processing of the Boltzman machine, the Liapunov function is occasionally forced to increase, so that the model is "bumped" out of local minima. This approach to overcoming local minima was proposed by Metropolis, Rosenbluth, Rosenbluth, Teller and Teller [13] in 1953 to characterize properties of thermodynamic systems. A series of state changes of the system are induced for which a small probability exists that a state change causes the Liapunov function of the system to increase. Later Kirpatrick, Gelatt and Vecchi[14] applied this approach, along with a schedule for the relative size of the probabilities, to optimization problems. Upon recognizing the first outer-product model as a solution to optimization problems, Ackley, Hinton and Sejnowski [15] applied Kirpatrick's work to to neural-network modeling. Modification of the outer-product model's decision function and adaptation function yielded the Boltzman machine.

3.2.1 Stability Analysis of Processing Dynamics

The Boltzman machine uses the Liapunov function of the first outer-product model with zero threshold levels and binary data-transfer values of (0,1). The weights between units are symmetrical as in the outer-product model, and the weights have zero value between units which are not in adjacent levels. Since the weights are symmetrical, only two weight matrices will be used in this analysis, wo and wh. A single hidden layer is assumed, but the analyses are analogous for many hidden layers. A unit in the output layer changes data-transfer value asynchronously from its current value with the probability:

$$p_{oi} = \frac{1}{1 + e^{-\Delta E_{oi}/T}}$$

T, a positive parameter referred to as temperature, governs the randomness of the changes in data-transfer values. The Liapunov function for the output layer, E_o, is similar to that of the first outer-product model:

$$E_o = -\sum_{i<j}\sum wo_{ji}\, x_{hj}\, x_{oi}$$

where wo_{ji} is the weight between the output unit i and the hidden unit j, x_{oi} is the data-transfer value of the output unit i, and x_{hj} is the data-transfer value of the hidden unit j. If the current data-transfer value of a unit is zero then the change in the Liapunov function for the data-transfer value changing from zero to one is:

$$\Delta E_{oi} = \sum_j wo_{ji}\, x_{hj}$$

so the probability of the data-transfer value of the unit changing to one is:

$$p_{oi} = \frac{1}{1 + e^{-\sum_j wo_{ji} x_{hj}/T}}$$

If the current data-transfer value of a unit is one then the change in the Liapunov function for the data-transfer value changing from one to zero is:

$$\Delta E_{oi} = -\sum_j wo_{ji}\, x_{hj}$$

so the probability of the data-transfer value of the unit remaining one is:

$$p_{io} = 1 - \frac{1}{1 + e^{\sum_j wo_{ji} x_{hj} / T}}$$

which reduces to

$$p_{io} = \frac{1}{1 + e^{-\sum_j wo_{ji} x_{hj} / T}}$$

Therefore, p_{oi} in the above form can be interpreted as the probability that the data-transfer value of output unit i will be one, regardless of its current value.

The probability that the data-transfer value is one for the units in the hidden layer and the output layer has the same form. For units in the hidden layer, the probability that unit i has a data-transfer value of one is:

$$p_{ih} = \frac{1}{1 + e^{-(\sum_j wo_{ji} x_{oj} + \sum_k wh_{ki} x_{ik})/T}}$$

Likewise for units in the input layer, the probability that unit i has a data-transfer value of one is:

$$p_{ii} = \frac{1}{1 + e^{-\sum_j wh_{ji} x_{hj} / T}}$$

The temperature parameter, T, is adjusted from a large number to a number close to zero according to a schedule which obeys the constraints outlined by Geman and Geman[16] . If T is large then data-transfer values are changed almost randomly. The model will require a long time to converge at a high temperature, but the chances of convergence to a local minima is small. Smaller temperatures cause the model to converge faster, but decreases the likelihood that the model will converge to the globally optimal solution. T equal to zero corresponds to the discrete outer-product model with T'_i equal to zero. When T equals zero, only the sign of the change in the Liapunov function, or the activation level in the discrete outer-product model, effects the data-transfer value. For any initial conditions and constant weights, the model will eventually converge to a stable set of data-transfer values for the units.

The stability of the processing dynamics in the Boltzman machine is easy to see from the Liapunov function, but has not yet been explicitly shown in the literature. Let the Liapunov function for the processing dynamics of the system be the sum of the Liapunov functions for each layer:

$$E = E_o + E_h + E_I$$
$$= -2 \sum_{i<j}\sum w_{oji}\, x_{hj}\, x_{oi} - 2 \sum_{k<j}\sum w_{hjk}\, x_{hj}\, x_{ik}$$

where E_h and E_I are the Liapunov functions for the hidden layer and the input layer, respectively. The Liapunov function for the system is bounded since all of the data-transfer values are either one or zero and the weights are constant. To show stability, condition 1.7c needs to be satisfied. For a change in the data-transfer value from zero to one in a unit, i, in the output layer,

$$\Delta E = -T \ln(\frac{1}{p_{oi}} - 1)$$

The change in the Liapunov function is always non-positive if $p_{oi} \leq 1/2$ assuming that T is kept constant during the change in the data-transfer value. If T changes slowly with respect to the change in the data-transfer values of the units, then the approximation that T is constant is reasonable. For a change in the data-transfer value from one to zero for unit, i, in the output layer,

$$\Delta E = -T \ln(\frac{1}{1 - p_{oi}} - 1) \qquad = -T \ln(\frac{p_{oi}}{1 - p_{oi}})$$

which is non-positive if $p_{oi} \leq 1/2$. The restriction that $p_{oi} \leq 1/2$ translates to the conservative restrictions on the weights that the values be positive to ensure that

$$\sum_j w_{oij}\, x_{hi} \geq 0$$

Analogous stability analyses for the other layers yield the result that all the changes in the overall Liapunov function are non-positive, so condition 1.7 of the second method of Liapunov is satisfied for this bounded Liapunov function. The conservative restriction which results from this analysis is that all values of the weights are positive for all of the layers. In addition, the assumptions are made that MIN = 0, MAX = 1, T_i = 0 and all the data-transfer values are binary (0,1). Any positive, slowly varying value of T will preserve stability (see table 3.1).

Table 3.1 Restrictions on Boltzman Machine

Connections	Weights and Adaptation	Decision Function
1) Multiple layers of units 2) Complete forward and backward inter-connection between layers 3) No lateral connections	1) Adaptation is performed before operation 2) $w_{ij} = w_{ji}$ 3) $w_{ii} = 0$ 4) η is small 5) Weights are positive	1) Threshold function is a probability equal to the sigmoid of $\sum_j w_{ji} x_j$ 2) MIN = 0, MAX = 1 3) All data-tranfer values are 0 or 1 4) T is positive and varies slowly

3.2.2 Stability Analysis of Adaptation Dynamics

The intent of the weight adaptation is to minimize the discrepancy between the distribution of the states of the units of the models and the distribution of the external input /desired output vector pairs presented to the model. A Liapunov function from information theory is used as a measure of this discrepancy. The Liapunov function is referred to as the asymmetric divergence or information gain:

$$G = \sum_s P(\underline{I}_s, \underline{d}_s) \ln \frac{P(\underline{I}_s, \underline{d}_s)}{P'(\underline{x}_i = \underline{I}_s, \underline{x}_o = \underline{d}_s)}$$

where $P(\underline{I}_s, \underline{d}_s)$ is the probability that the the external input / desired output pair, $(\underline{I}_s, \underline{d}_s)$, occurs. $P'(\underline{x}_i = \underline{I}_s, \underline{x}_o = \underline{d}_s)$ is the probability that the data-transfer values of the input units are $\underline{I}_s$ and the data-transfer values of the output units are $\underline{d}_s$ when no external vectors are applied to the model. G is positive when

$$P(\underline{I}_s, \underline{d}_s) > P'(\underline{x}_i = \underline{I}_s, \underline{x}_o = \underline{d}_s) \tag{3.1}$$

for at least one pattern, and G equals zero when the above comparison is equal for all patterns, s. Theoretically, the equation 3.1 is always true, but care must be taken with the statistics to ensure that this equation is true. Assuming that these precautions are taken, G is a conventional Liapunov function and its derivative should be negative to fulfill condition 1.2 of the second method of Liapunov.

To minimize the value of G, a steepest descent method is used for the adaptation, where the change in the weights is proportional to the negative of the derivative of G with respect to the weights currently being adapted:

$$\begin{aligned} \frac{dG}{dt} &= \frac{\partial w_{ji}}{\partial t} \frac{\partial G}{\partial w_{ji}} \\ &= -\left(\frac{\partial w_{ji}}{\partial t}\right)^2 \end{aligned}$$

where w_{ji} is either wo_{ij} or wh_{ji} and

$$\frac{\partial w_{ji}}{\partial t} = -\frac{\partial G}{\partial w_{ji}}$$

This derivative satisfies condition 1.5c for Liapunov's second method, so the adaptation is asymptotically stable. The adaptation function is derived from the derivative of G with respect to the weights. Since $P(\underline{I}_s, \underline{d}_s)$ is constant with respect to changes in the weights,

$$\frac{\partial G}{\partial w_{ij}} = \sum_s \frac{P(\underline{I}_s, \underline{d}_s)}{P'(\underline{x}_i = \underline{I}_s, \underline{x}_o = \underline{d}_s)} \frac{\partial P'(\underline{x}_i = \underline{I}_s, \underline{x}_o = \underline{d}_s)}{\partial w_{ji}}$$

To derive the adaptation function first P' is presented in terms of the values of the weights, then the Boltzman property is used to calculate the derivative of P' with respect to the weight being adapted.

The order of the units is unimportant, so the input and output units are re-ordered so that for a given external input / desired output pair, s, the ones are desired for the first K output units and the first L input units. The probability of this pattern for a given state, α, of the hidden units when no external vectors are applied for n_o units in the output layer and n_i units in the input layer is:

$$P'(\underline{x}_i = \underline{I}_s, \underline{x}_o = \underline{d}_s, \underline{x}_\alpha) = \prod_{k=1}^{K} p_{\alpha ok} \prod_{kk=K}^{n} (1-p_{\alpha okk}) \prod_{l=1}^{L} p_{\alpha il} \prod_{ll=L}^{n} (1-p_{\alpha ill})$$

$$= \frac{1}{\prod_{k=1}^{K} \left[1 + e^{-\sum_j wo_{kj}\, x_{\alpha hjs}/T}\right]} \prod_{kk=K}^{n_o} \left[1 - \frac{1}{\left[1 + e^{-\sum_j wo_{kkj}\, x_{\alpha hjs}/T}\right]}\right]$$

$$\times \frac{1}{\prod_{l=1}^{L}\left[1+e^{-\sum_j wh_{lj}\, x_{\alpha hjs}/T}\right]} \prod_{ll=L}^{n_i}\left[1-\frac{1}{\left[1+e^{-\sum_j wh_{llj}\, x_{\alpha hjs}/T}\right]}\right]$$

$\underline{x}_{\alpha h}$

is the final data transfer value of the jth hidden unit when the input and output units give the response, $(\underline{I}_s, \underline{d}_s)$. wo_{kj} is the value of the weight between the jth hidden unit and the kth output unit, and wh_{lj} is the value of the weight between the jth hidden unit and the lth input unit. This equation can be manipulated into the following form since x_{oks} and x_{ils} will be either one or zero:

$$P'(\underline{x}_i = \underline{I}_s, \underline{x}_o = \underline{d}_s, \underline{x}_{\alpha h}) = \frac{e^{(-\sum_k \sum_j wo_{kj}\, x_{\alpha hjs}\, x_{\alpha oks} - \sum_{l=1}^{L} \sum_j wh_{lj}\, x_{\alpha hjs}\, x_{\alpha ils})/T}}{Z_\alpha}$$

$$= \frac{e^{-E_{\alpha s}/T}}{Z_\alpha}$$

where $1/Z_{\alpha s}$ is the probability that all of the input and output data-transfer values are zero when no external influence exists. This value does not depend on the pattern being presented.

$$Z_\alpha = \prod_{k=1}^{n_o}(1+e^{-\Delta E_{\alpha o}/T}) \prod_{l=1}^{n_i}(1+e^{-\Delta E_{\alpha i}/T})$$

Finally, the dependence of P' on w_{ij} is:

$$P'(\underline{x}_i = \underline{I}_s, \underline{x}_o = \underline{d}_s) = \sum_\alpha \frac{e^{-E_{\alpha s}/T}}{Z_\alpha}$$

$$= \sum_{\alpha} \frac{e^{(-\sum_k \sum_j wo_{kj}\, x_{\alpha hjs}\, x_{\alpha oks} - \sum_{l=1}^{L} \sum_j wh_{lj}\, x_{\alpha hjs}\, x_{\alpha ils})/T}}{Z_{\alpha}}$$

The derivative of Z_s would be difficult to find so the Boltzman property is employed to simplify the calculation. The Boltzman distribution is the ratio of two probabilities with the same base probability, 1/Z.

$$\frac{P_a}{P_b} = e^{-(E_a - E_b)}$$

In this case, the Boltzman distribution is identical to the original distribution since the denominator in the following expression is one if all of the possible patterns have been learned:

$$\frac{P'(\underline{x}_i = \underline{I}_s, \underline{x}_o = \underline{d}_s, \underline{x}_{\alpha h})}{\sum_s P'(\underline{x}_i = \underline{I}_s, \underline{x}_o = \underline{d}_s, \underline{x}_{\alpha h})} = \frac{\sum_{\alpha} e^{-E_{\alpha s}/T}}{\sum_{\alpha} \sum_s e^{-E_{\alpha s}/T}}$$

Now the derivative can be calculated for an output weight w_{oji}. The results for a hidden weight would be analogous:

$$\frac{\partial P'(\underline{x}_i = \underline{I}_s, \underline{x}_o = \underline{d}_s)}{\partial w_{ji}} = \frac{1}{T} \sum_{\alpha} \left[\frac{x_{\alpha hjs} x_{\alpha his}\, e^{-E_{\alpha s}/T}}{\sum_s e^{-E_{\alpha s}/T}} \right.$$

$$\left. - \frac{e^{-E_{\alpha s}/T}}{(\sum_s e^{-E_{\alpha s}/T})^2} \sum_s e^{-E_{\alpha s}/T} x_{\alpha hjs} x_{\alpha ois} \right]$$

This derivative can then be expressed in terms of P':

$$-P'(\underline{x}_i = \underline{I}_s, \underline{x}_o = \underline{d}_s) \sum_s P'(\underline{x}_i = \underline{I}_s, \underline{x}_o = \underline{d}_s, \underline{x}_{\alpha h}) x_{\alpha hjs} x_{\alpha his})]$$

Substituting this result back into the original derivation of the derivative of G:

$$\frac{\partial G}{\partial wo_{ij}} = \frac{1}{T} [\sum_s \frac{P(\underline{I}_s, \underline{d}_s)}{P'(\underline{x}_i = \underline{I}_s, \underline{x}_o = \underline{d}_s)} \sum_\alpha (x_{\alpha hjs} x_{\alpha his} P'(\underline{x}_i = \underline{I}_s, \underline{x}_o = \underline{d}_s, \underline{x}_{\alpha h})$$
$$- \sum_s P(\underline{I}_s, \underline{d}_s) \sum_s P'(\underline{x}_i = \underline{I}_s, \underline{x}_o = \underline{d}_s, \underline{x}_{\alpha h}) x_{\alpha hjs} x_{\alpha his})]$$

Since all possible patterns are presented,

$$\sum_s P(\underline{I}_s, \underline{d}_s) = 1$$

Further simplification is made by defining p'_{oij} to be the probability that the data units x_{oi} and x_{hj} both have data-transfer values of one.

$$p'_{oij} = \sum_s \sum_\alpha x_{\alpha hjs} x_{\alpha his} P'(\underline{x}_i = \underline{I}_s, \underline{x}_o = \underline{d}_s, \underline{x}_{\alpha h})$$

Bayes' theorem is used to simplify the first term in the derivative:

$$\frac{P'(\underline{x}_i = \underline{I}_s, \underline{x}_o = \underline{d}_s, \underline{x}_{\alpha h})}{P'(\underline{x}_i = \underline{I}_s, \underline{x}_o = \underline{d}_s)} = P'(\underline{x}_h \mid \underline{x}_i = \underline{I}_s, \underline{x}_o = \underline{d}_s)$$

Then the definition of the probability p_{oij} is made. p_{oij} is the probability that units x_{oi} and x_{hj} both have data-transfer values of one when the input units are forced to the values of the external input vector and the output units are forced to the values of the desired output vector.

$$p_{oij} = \sum_s \sum_\alpha x_{\alpha hjs} x_{\alpha his} P(\underline{I}_s, \underline{d}_s) P'(\underline{x}_h \mid \underline{x}_i = \underline{I}_s, \underline{x}_o = \underline{d}_s, \underline{x}_{\alpha h})$$

Finally, after all of this derivation, most of which is not published in as much detail in the literature, a simple form for the derivative of G is found:

$$\frac{\partial G}{\partial w_{ij}} = -\frac{1}{T}\,(p_{o1ij} - p'_{oij})$$

These values of p'_{oij} and p_{oij} are obtained statistically by observing the behavior of the system with and without external input / desired output vector pairs and noting when the units converge to a data-transfer value of one instead of a data-transfer value of zero.

Now that the derivative of the weight with respect to time has been derived which satisfies condition 1.2 for stability, the approximation is made that the change in weights is discrete. If the weight changes are small enough relative to the function G for a specific application, then stability will still hold. The adaptation function for the weights for the output layer is:

$$\Delta wo_{ji} = \eta\,(p_{oij} - p'_{oij})$$

for the output weights and by an analogous derivation, the adaptation function for the weights for the hidden layer is:

$$\Delta wh_{ji} = \eta\,(p_{hij} - p'_{hij})$$

where η is some parameter dependent on the inverse of T and p and p' are statistics.

The assumptions involved in the analysis of the stability of the adaptation dynamics are that the sum of the probabilities of the inputs, $(\underline{I}_s, \underline{d}_s)$ is one, and that only one weight is changed at a given time. The first assumption is valid if the inputs are repeated frequently enough for the statistics to be good approximations to the actual probabilities. This restriction should also ensure that $P' < \mathrm{P}$, although this condition is difficult to test. The restriction that η is small makes the approximation that only one weight changes at a time reasonable. No provisions are made to prevent the values of the weights from being negative. Negative weights could cause instability in the processing, so the weights need to be checked after adaptation. The adaptation does ensure that the weights are symmetrical.

3.2.3 Summary of Stability Analyses

In the systems implemented by Ackley, et al, relative size of the probabilities were ignored, so the amount of the weight adaptation is constant. The value of the weight is decremented if $p_{ij} < p'_{ij}$ and incremented if the sizes of the probabilities are reversed. No change occurs if the probabilities are the same. This approach is a good approximation if η is small, and the approach alleviates some of the effects of inaccurate statistics. The probabilities for adaptation are computed at a temperature which has an value of the temperatures used for processing. First the external input vector is applied and the data-transfer values of the output units are set to the desired vector. Some noise is introduced in this process to keep the probability from becoming too small for other possible external input vectors. The resulting model is allowed to reach equilibrium twice and the probabilities for all of the weights are recorded. The next step is to remove the external input vector and the desired output vector to add to the statistic for p' for all of the weights. By using noisy external input vectors to train the network, the weights will almost always converge.

The Liapunov functions for the Boltzman machine are fairly simple, but the resulting neural-network model is very difficult and slow to implement. Stochastic processing theoretically avoids sub-optimal processing and weight adaptation, but at an enormous expense in terms of processing time and uncertainty in statistical parameters. The adaptive resonance theory seeks the same result of a neural-network model with both processing and adaptation dynamics, but at lower expense in terms of processing time and adaptation time. The penalty paid is a much less elegant model in terms of the Liapunov functions, than the Boltzman machine.

Chapter 4
ANALYSIS OF ADAPTIVE RESONANCE THEORY

4.1 Overview of Adaptive Resonance Theory

The dynamics of adaptive resonance theory are the most complex of the four models analyzed, but the Liapunov functions employed are the simplest [17] . The most complex aspect of adaptive resonance theory is the determination of what units and parameters are a source of dynamics at a given time. Unlike those in the outer-product model and the Boltzman machine, the connections between units in adaptive resonance theory do not always carry data-transfer values. The dynamics of the model are switched from layer to layer. For each layer, a different set of active connections, weights, decision function and Liapunov functions for processing and adaptation are employed.

The complexity of the model is limited somewhat by only using two layers of units, an input layer and an output layer (see figure 4.1). The feed-forward connections associate patterns of data-transfer values in the input layer (F1) to a unit in the output layer (F2). The feedback connections send the pattern which the selected unit in F2 was previously taught to F1 for comparison with the external input vector. Lateral connections exist between units in the F2 layer. The lateral connections are responsible for the dynamic processing of data-transfer values. The feed-forward and feedback connections are each used in different processing phases so the units do not interact continuously as in the outer-product model and the Boltzman machine.

The units interact between layers through the adaptation of the weights. The values of the weights reflect the pattern which the output unit has been adapted to recognize. The feed-forward weights are normalized so that the unit(s) which matches the most elements of the external input vector has the largest activation level. The feedback weights send the vector which the unit with the largest activation level in F2 was taught to recognize to the F1 layer. The criterion for instigating adaptation of the feed-forward and feed-back weights is the output of the arousal unit. The arousal unit can either cause the weights to adapt, or the unit can cause a different F2 unit to have the largest activation level. The action performed depends upon a comparison of a constant, referred to as the vigilance, to the ratio of the norm of the product of the two patterns to the pattern on F1. The adaptation of the weights is performed differently for the two levels so that the different types of information are encoded. The lateral weights in the layers are kept constant. The values of the lateral weights are pre-set depending on the number of units in the layer. In the F1 layer, the lateral weights are set to create an upper bound on the activation level of the F1 units. In the F2 layer, the lateral weights are set to maximize the likelihood that only one unit has a positive output.

In both the continuous outer-product model and adaptive resonance theory, the current activation level is based on the previous activation level. The difference is that the activation level in adaptive resonance theory is determined by a second degree equation as opposed to the first degree

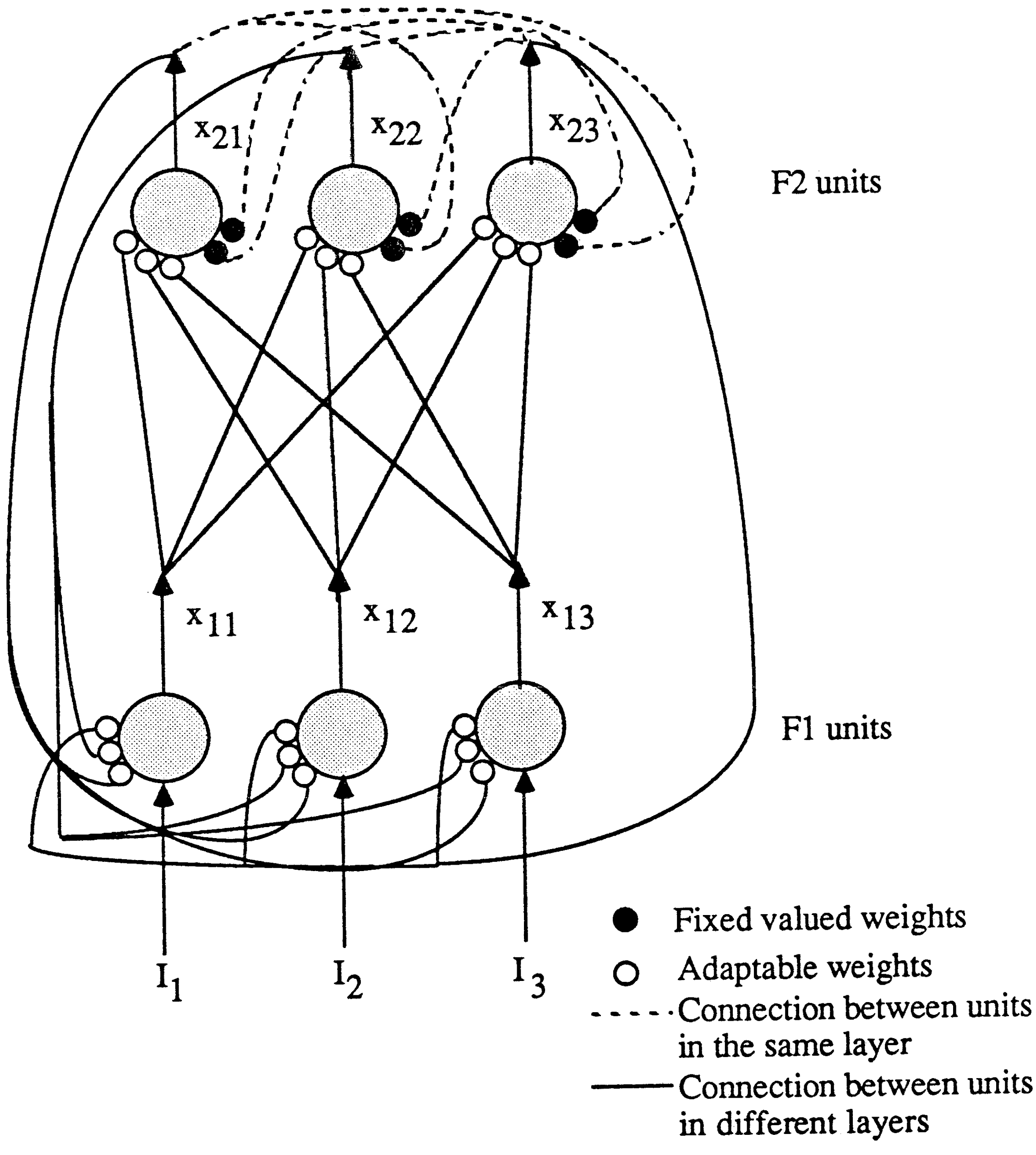


Figure 4.1 Functional Components of Adaptive Resonance Theory

equation used in the continuous outer- product model. The output of a unit is determined by different threshold functions for units in different layers. The units in the F2 layer generally have a threshold function which is close to linear. The data-transfer value is limited more by the activation function than the threshold function. Any thresholding function can be used for the units in the Fl layer. The data-transfer value for these units is bounded mostly by the values of the external input vector and the output data-transfer values of the units in the F2 layer. The threshold function plays a much smaller role in adaptive resonance theory than in other neural-network model.

Five separate phases are involved in the processing of each external input vector as opposed to the two phases, adaptation and processing, in the Boltzman machine. Two of the five phases in adaptive resonance theory involve the adaptation of the weights. The other three phases involve the processing dynamics of the units of one of the layers. Each phase requires a different set of restrictions and assumptions which are difficult to ascertain from the literature. Despite adaptive resonance theory's considerable success in simulating psychological behavior, the exact specifications for the parameters of the model are not obvious. Under reasonable assumptions about the model, all five phases are shown internally stable in section 4.2, although stability of the model as a whole depends upon the application.

4.2 Analysis of Stability of Adaptive Resonance Theory

Adaptive resonance theory is the result of several decades of work in mathematical psychology to attempt to model psychological phenomena. Most analyses of adaptive resonance theory are based on the assumption that the data-transfer values and the weights change continuously within each phase of operation. Adaptive resonance theory is a combination of discrete and continuous interactions. Each discrete phase contains continuous dynamics which can be shown to converge to a solution for either the data-transfer values or the weights. This solution is then passed in a discrete step as input to the next phase. This approximation allows each phase to be analyzed independently assuming that the continuous processes either converge in a finite amount of time or are approximated with the final results. The five phases are as follows:

Phase 1: (F1 data-transfer values) The external input vector is processed by the F1 units.

Phase 2: (F2 data-transfer values) A characteristic F2 unit (or units) is chosen based on the weighted data-transfer values from the F1 units.

Phase 3: (F1 data-transfer values) Both the weighted data-transfer values from the F2 units and the external input vector are processed by the F1 units. The arousal unit uses the values of the F1 units to determine whether to proceed to Phase 4 or to reset the F2 layer and repeat Phases 1-3.

Phase 4: (F2 weights) The bottom-up weights between layers F1 and F2 are adapted.

Phase 5: (F1 weights) The top-down weights between layers F1 and F2 are adapted.

These phases are repeated for each external input vector. The model is stable if the values of the weights do not oscillate for different sequences of external input vectors. The model is asymptotically stable for a fixed set of external input vectors if the changes in the weights approach zero.

4.2.1 Stability Analysis of Phase 1

The stability of the processing in the first phase is determined by the dynamics of the equations for the F1 units. In general the equation for the i^{th} unit in the F1 layer and the j^{th} units in the F2 layer is:

$$\varepsilon \frac{du_{1i}}{dt} = -u_{1i}(t) + (1 - A_1 u_{1i}(t))\,(I_i + D_1 \sum_j x_{2j}\, wt_{ji}) - (B_i + C_1 u_{1i}(t)) \sum_j x_{2j}$$

where $A_1, D_1, B_1,$ and C_1 are all positive constants, and wt_{ji} is the top-down weight between unit i in F1 and unit j in F2. Since the units of F2 are not yet activated, $x_{2j} = 0$. The equation can be reduced to:

$$\varepsilon \frac{du_{1i}}{dt} = -(1 + A_1 I_i)\, u_{1i}(t) + I_i \tag{4.1}$$

If the unit converges ($du_{1i}/\mathrm{dt} = 0$), then the activation level for unit i will be:

$$u_{1i,eq} = \frac{I_i}{1 + A_1 I_i}$$

which is always positive. The model will reach this value for the activation level of the F1 units in this phase if the initial activation level is zero, the activation levels are continuous, and du_i/dt > 0 for all activation levels less than the final value. The Liapunov function is

$$E = \sum_i \mid u_{1i} - \frac{I_i}{1 + A_1 I_i} \mid$$

which is positive in the form of conditions 1.2. The derivative is:

$$\frac{dE}{dt} = -\sum_i \frac{du_{1i}}{dt}$$

when for all units, i:

$$u_i < \frac{I_i}{1 + A_1 I_i}$$

Therefore, $\frac{du_{1i}}{dt} > 0$ for all i satisfies condition 1.2c for Liapunov's second method. Algebraic manipulation of the equation 4.1 shows that du_i/dt ≥ 0 implies that

$$u_{1i}(t) \leq \frac{I_i}{1 + A_1 I_i}$$

which is true for all positive values of A_1 when $u_{1i}(0)$ equals zero. The data-transfer value for each unit is some function, f_1, of the activation level. The only requirement on the threshold function, f_1, is monotonicity which will make the data-transfer value stable when the activation level is stable. The threshold function does not affect processing dynamics in this phase. Therefore, the restrictions are monotonicity of f_1 and $u_i(0) = 0$ for all F1 units.

The intent of the first phase is for the data-transfer values of the units of F1 to be proportional to the external input vector. This condition will hold if A_1 is much smaller than the inverse of value of the external input, i.e. $A_1 I_i$ << 1 for all i. Otherwise, the activation level is subject to saturation to the value one.

4.2.2 Stability Analysis of Phase 2

The second phase is the convergence of the units in the F2 level. The equation for the dynamics of the activation levels, u_{2j}, of the F2 units is:

$$\varepsilon \frac{du_{2j}}{dt} = -u_{2j}(t) + (1 - A_2 u_{2j}(t))\,(x_{2j}(t) + D_2 \sum_i x_{1i}\, wb_{ij}) - (B_2 + C_2 u_{2j}(t)) \sum_{k \neq j} x_{2k}(t)$$

where $A_2, D_2, B_2,$ and C_2 are all positive constant parameters, and x_{2k} is the data-transfer value of another unit of F2, and wb_{ij} is the bottom-up weight between unit j in F2 and unit i in F1. The weights and the data-transfer values from the F1 units, x_{1i}, are assumed constant for this phase. Descriptions of adaptive resonance theory[18] , refer to several articles for the choice of parameters for the F2 units. Unfortunately, none of the equations in these articles exactly correspond to the equation cited for adaptive resonance theory. Notationally, the most similar equation for a network of units appears in the paper by Ellias and Grossberg[19] :

$$\frac{du_{2j}}{dt} = -A\ u_{2j}(t) + (B - x_{2j}(t))\,(\sum_k x_{2k}(t)\,C_{kj} + I_j) - u_{2j}(t)\sum_k f\,(y_{2k}(t))\,D_{kj}$$
$$- u_{2j}(t)\sum_k f\,(y_{2k}(t))\,D_{ki}$$

$$\frac{dy_{2j}}{dt} = -E\ y_{2j}(t) + \sum_k u_{2k}(t)\,F_{kj}$$

where

$$A = \frac{1}{\varepsilon}$$

$$B = \frac{1}{A_2\,\varepsilon}$$

$$C_{ki} = A_2$$

$$I_i = A_2\,D_2 \sum_i x_{1i}\ wb_{ij}$$

$$D_{ki} = E = F_{kj} = 0$$

with restrictions that $B_2 = -1$ and $C_2 = A_2$. This choice of equations has been shown to demonstrate limit cycles and many areas of instability so this equation is not very appropriate for adaptive resonance theory. By placing more restrictions on the model, the equation can be made similar to an equation discussed in an earlier work by Grossberg[20] .

$$\frac{du_{2j}}{dt} = -\left(A + \sum_{k \neq j} x'_{2k}(t)\right) u_{2j}(t) + (B - u_{2j}(t))\, u_{2j}(t)$$

where

$$A = \frac{1}{\varepsilon}$$

$$B = \frac{1}{A_2}$$

$$x'_{2k}(t) = \frac{A_2}{\varepsilon}\, x_{2k}(t)$$

with the restrictions that $B_2 = 0$ and $C_2 = 1$, and the initial conditions:

$$u_{2j}(0) = D_2 \sum_i x_{1i}\, wb_{ij}$$

This approximation has many of the interesting dynamics which are alluded to in the text of the adaptive resonance description. Since the equation only depends on its own activation level and the sum of all of the other activation levels as variables, the model preserves the relative sizes of the data-transfer values among the units in F2. Once the activation levels of two units are equal, the values will remain equal. A model based on this equation with a threshold function, f_2 which satisfies the following:

$$\frac{f_2(u_{2j}(t))}{u_{2j}(t)} \geq \frac{A_2}{\varepsilon}$$

can also be shown to converge to either a state in which all of the units in F2 have activation levels which are at one of their two bounds (0 or 1), or a state in which all of the units in F2 have the same activation level. This property can be shown using the following bounded function:

$$E = max(U_{2j}(t)) - min(U_{2j}(t))$$

where

$$U_{2j}(t) = \frac{u_{2j}(t)}{\sum_k u_{2k}(t)}$$

E is bounded since u_{2j} is bounded between zero and $1/A_2$. If the initial state of the network is

$$max(U_{2j}(t)) = min(U_{2j}(t)) = \frac{1}{n}$$

then the value of the Liapunov function is zero and this state of the units is stable. Otherwise, the equation of the activation level can be rewritten in the form:

$$\frac{dU_{2j}}{dt} = \frac{1}{A_2} U_{2j}(t) \sum_k U_{2k}(t) \left(\frac{x_{2j}(t)}{u_{2j}(t)} - \frac{x_{2k}(t)}{u_{2k}(t)}\right)$$

The derivative of max($U_{2j}(t)$) will be greater than zero and the derivative of min($U_{2j}(t)$) will be less than zero if $f_2(u_{2j}(t))/u_{2j}(t)$ is convex, continuous and monotonically increasing with the value of $f_2(u_j(t))/u_j(t)$ at $u_j(t) = 0$ greater than or equal to A_2/ε. These conditions cause the derivative of the Liapunov function to always be greater than zero. Thus, the conditions 1.3 for Liapunov's second method are satisfied and the phases is stable. The parameters, ε and A_2, impact the convergence rate. Large values of A_2 will cause faster convergence, but the width of the dynamic range will be smaller. The maximum size of A_2 is implementation dependent. Small values of ε can be used in a discrete approximation to smooth the convergence process. This phase will converge to a stable state in which the unit or units with non-zero activation levels are referred to as the characteristic F2 unit(s).

4.2.3 Stability Analysis of Phase 3

The third phase reprocesses the data-transfer values of the F1 units based on the data-transfer values of the F2 units as determined in phase 2. The initial values for this phase are the final values for the F1 units in the first phase. Recall that the the equation for the i^{th} unit in the F1 layer and the j^{th} units in the F2 layer is:

$$\varepsilon \frac{du_{1i}}{dt} = -u_{1i}(t) + (1 - A_1 u_{1i}(t))\,(I_i + D_1 \sum_j wt_{ji}\, x_{2j}(t))$$
$$- (B_i + C_1 u_{1i}(t)) \sum_j x_{2j}(t)$$

where $A_1, D_1, B_1,$ and C_1 are all positive constants, and wt_{ji} is the top-down weight between unit i in F1 and unit j in F2. The data-transfer values from the F2 units, x_{2j}, and the weights, wt_{ji} are assumed to be constant in this phase. If the units converge, then the final activation level for this phase will be:

$$u_{1i,eq} = \frac{I_i + D_1 \sum_J wt_{Ji} + B_1}{1 + A_1 I_i + A_1 D_1 \sum_J wt_{Ji} + C_1 \sum_J x_{2J}}$$

where wt_{Ji} is the weight between the F1 unit i and the unit(s) J in F2 which has a data-transfer value of one.

The positive Liapunov function is:

$$E = \sum_i \left| u_{1i} - \frac{I_i + D_1 \sum_J wt_{Ji} + B_1}{1 + A_1 I_i + A_1 D_1 \sum_J wt_{Ji} + C_1 \sum_J x_{2J}} \right|$$

and its derivative is:

$$\frac{dE}{dt} = \sum_i \frac{du_{1i}}{dt}$$

when all activation levels of units i are:

$$u_{1i,eq} < \frac{I_i + D_1 \sum_J wt_{Ji} + B_1}{1 + A_1 I_i + A_1 D_1 \sum_J wt_{Ji} + C_1 \sum_J x_{2J}}$$

and

$$\frac{dE}{dt} = -\sum_i \frac{du_{1i}}{dt}$$

when all activation levels of units i are:

$$u_{1i,eq} > \frac{I_i + D_1 \sum_J wt_{Ji} + B_1}{1 + A_1 I_i + A_1 D_1 \sum_J wt_{Ji} + C_1 \sum_J x_{2J}}$$

Showing that the derivative is negative if the initial activation level is greater than the desired final value and positive if the initial activation level is less than the desired final activation level is sufficient to show that the activation level converges to the desired final value since condition 1.2c is satisfied. The derivative of the activation level for the initial state of the activation level at the final value from phase 1 is:

$$\frac{du_{1i}(0)}{dt} = \frac{D_1 \sum_J wt_{Ji} - (A_1 B_1 + C_1 \sum_J x_{2J}) I_i}{1 + A_1 I_i} \tag{4.2}$$

If the desired final activation level is greater than the initial activation level:

$$\frac{I_i + D_1 \sum_J wt_{Ji} + B_1}{1 + A_1 I_i + A_1 D_1 \sum_J wt_{Ji} + C_1 \sum_J x_{2J}} > \frac{I_i}{1 + A_1 I_i}$$

then this condition implies that

$$D_1 \sum_J wt_{Ji} > (C_1 \sum_J x_{2J} + A_1 B_1) I_i$$

which makes the derivative in equation 4.2 positive assuming $A_1 I_i > -1$. Likewise when the desired activation level is less than the initial activation level, the derivation is negative. Therefore, the conditions 1.2 of Liapunov's second method are satisfied. This phase converges to a set of data-transfer values for the units in F1 if the thresholding function is monotonically increasing or decreasing. The restriction that $A_1 I_i <-1$ is consistent with the restriction that $A_1 I_i \ll 1$ from phase 1.

If the arousal unit resets the positive units in F2, the preceding phases will be repeated with fewer units active in F2. Since the convergence in each of these phases is not dependent upon the number of units in F2, the model will remain stable. If the arousal unit does not reset the units in F2 then the values of the weights will be adapted. The weight adaptation determines the model's stability over varying external input vectors, since all of the other phases will converge for any external input vector. The model is stable if after some finite time, an external input vector always causes the model to respond in the same manner, ie. causes the same unit(s) in F2 to converge to positive activation level.

4.2.4 Stability Analysis of Phase 4

The bottom-up weights record the relative importance of each input to the activation of the corresponding F2 unit with respect to the total number of positive inputs. The literature describes the weight changes in a continuous form, but the weight changes are usually thought of as discrete due to the discrete nature of simulation and many potential implementations. The "fast-learning" adaptation is the approach suggested where the weights are adapted to have the value of the steady-state solution of the weight equations.

The adaptation of the bottom-up weights is characterized by the following equations.

$$\frac{dwb_{ij}}{dt} = K\ x_{2j}\ [(1 - wb_{ij}(t))\, L\ x_{1i} - wb_{ij} \sum_{l \neq i} x_{1l}]$$

where the data-transfer values of all of the units are constant for this phase. For fast learning:

$$wb_{ij}(t) = \frac{L\ x_{1i}}{(L - 1)\, x_{1i} + \sum_{l} x_{1l}}$$

The fast learning response is the value of wb_{ij} when the derivative of the adaptation function is zero. When the weight is greater than this value in the continuous learning case, then the derivative is negative. Likewise the adaptation approaches the steady state value when the weight is less than this value since the derivative is then positive. If L is close to one then the weights are approximately equal to the normalized value of x_i and the sum of the weights connected to each unit in F2 is L. The parameter K controls the speed of the adaptation.

Generalization of patterns can be performed by not allowing the adaptation to reach steady-state for any one external input vector. In a discrete system, a fraction of the fast-learning value could be added to the weight at the adaptation phase for each different external input vector. The arousal level affects the similarity between the external input vectors which are allowed to adapt the weights of a unit in F2. This method of implementing generalization in the patterns encoded by the weights is also applicable to the top-down weights.

4.2.5 Stability Analysis of Phase 5

The top-down weights record the pattern which caused the unit to be one of the units with a data-transfer value of one at the end of phase 2. The equations for adaptation of top-down weights more closely resemble the typical learning equations:

$$\frac{dwt_{ji}}{dt} = x_{2j} \left(-wt_{ji}(t) + x_{1i} \right)$$

where the data-transfer values of all the units are constant. The fast learning equation is:

$$wt_{ji}(t) = \frac{x_{1i}}{x_{2j}}$$

As for the bottom-up weights, the fast learning value is the steady-state solution of the continuous adaptation equation. The derivative of the weights will be positive if the weight is below the steady-state value and negative if the weight is above the steady-state value. The weights will equal the output values of the F1 units if the corresponding F2 unit is active. The weights can be made a composite of the patterns which activate an F2 unit by the methods for generalization mentioned in section 4.2.4.

4.2.6 Summary of Stability Analysis

The values of both the bottom-up and top-down weights will oscillate for different external input vectors if the arousal unit does not ensure that only similar external input vectors activate a unit in F2. The arousal unit resets a unit if

$$\frac{\sum_J \sum_i I_i \, wt_{Ji} \, x_{2J}}{\sum_i I_i} < \rho$$

where ρ is the arousal level (or vigilance parameter) approximately equal to one. Since $x_{1i} \approx I_i$, the numerator is approximately equal to the number of weights, which in turn, are equal to x_i. If most of the weights are not equal, then the ratio will be less that ρ and the arousal unit will cause the F2 unit to be reset. The weights on units which have not been adapted must be uniform enough that the ratio will be greater than ρ if the unit is activated. This will cause new patterns to be adapted to units in F2 which have not previously had their weights adapted. The closer ρ is to one, the fewer times each weight will be adapted, which will minimize the oscillations in weights, which in turn minimizes the instability in the model for a set of external input vectors. The trade-off is that less generalization of patterns is performed in more stable models.

In general, adaptive resonance theory is extremely stable and requires very few restrictions on the parameters of the model (see table 4.1). Most of the restrictions are due to the phase 2 computation of the state of the F2 units to arrive at only one unit with a data-transfer value equal to one. The Center for Adaptive Systems in Boston, MA has done extensive research on the dynamics of different networks of units. The adaptive resonance theory model is the most complete model to arise from this work. The type of applications which adaptive resonance theory is applied to are usually auto-association or generalization. Generalization is particularly appropriate since modification of the vigilance parameter adjusts the size of the partitions between patterns.

Table 4.1 Restrictions on Adaptive Resonance Theory

Connections	Weights and Adaptation	Decision Function
1) Two layers of units	1) Adaptation is only occurs after the correct F2 unit has been selected	1) Transfer function for F1 units is monotone increasing or decreasing
2) Feed-forward and feedback connections between layers	3) $\rho < 1$	2) Transfer function for F2 units has a time derivative which is convex, continuous and monotone increasing and $f_2(0) > A_2/\varepsilon$
3) No connection between units in F1 layer	4) $L < 1$	4) $-1 < A_1 I_i << 1$
4) Lateral connection between units in F2 layer		5) $A_2 > 0$
		6) $B_2 = 0$
		7) $C_2 = 1$
		8) $D_2 > 0$

Chapter 5
ANALYSIS OF BACKPROPAGATION MODEL

5.1 Overview of Backpropagation

The backpropagation model is an example of a neural-network model with dynamics determined only by the adaptation function. In the backpropagation model, the feedback, which is the source of the model's dynamics, is between the user and the model. This form of feedback differs from the feedback between units used as the source of the dynamics in the other three models. The operation of the backpropagation model without adaptation is feed-forward, with no dynamics. Another difference between the backpropagation model and the other neural-network models analyzed here is that a discrete event in the dynamics of the backpropagation model is the presentation of an external input vector and the associated desired output vector. For the discrete outer-product model and the Boltzman machine, a discrete event is the change in the data-transfer value of one unit. For adaptive resonance theory, a discrete event is the change in the state of a layer of units. As with the Boltzman machine, the best applications for the backpropagation model are those that require hetero-association of two vectors and have data in the form of an external input/desired output vector pair.

The functional components of the backpropagation model are the simplest to implement of the four neural-network models analyzed here. The form of the network is multiple layers of units with feed-forward connections to units in the succeeding layer. Units are not connected to the other units in the layer (see figure 5.1). The first layer is the input layer, but since its weights are not adaptable, the data transfer values for the input units are represented by the values of the external input vector. The layer(s) between the input layer and the output layer are the hidden layer(s) which provides intermediary representations of the information which the model is processing. The last layer is the output layer which represents the solution to the user. To adapt the values of the weights, an error is generated from the difference between the data-transfer values of the output units and the associated desired output values. A user provides the desired output vector along with the external input vector during the adaptation process. The decision function is not constrained by the model to any particular form other than to have a bounded threshold function which is monotonically increasing. The sum of the weighted inputs is the activation level, and usually a sigmoidal function is used for thresholding.

The analysis which follows rejects the assumptions of continuous processing made in the original backpropagation stability analysis by Rumelhart and McClelland[21] , and shows that stability can be shown for discrete processing.

5.2 Analysis of Stability of the Backpropagation Model

The Liapunov function for the backpropagation model is the same as the error function used by other least-squares algorithms. The Liapunov func-

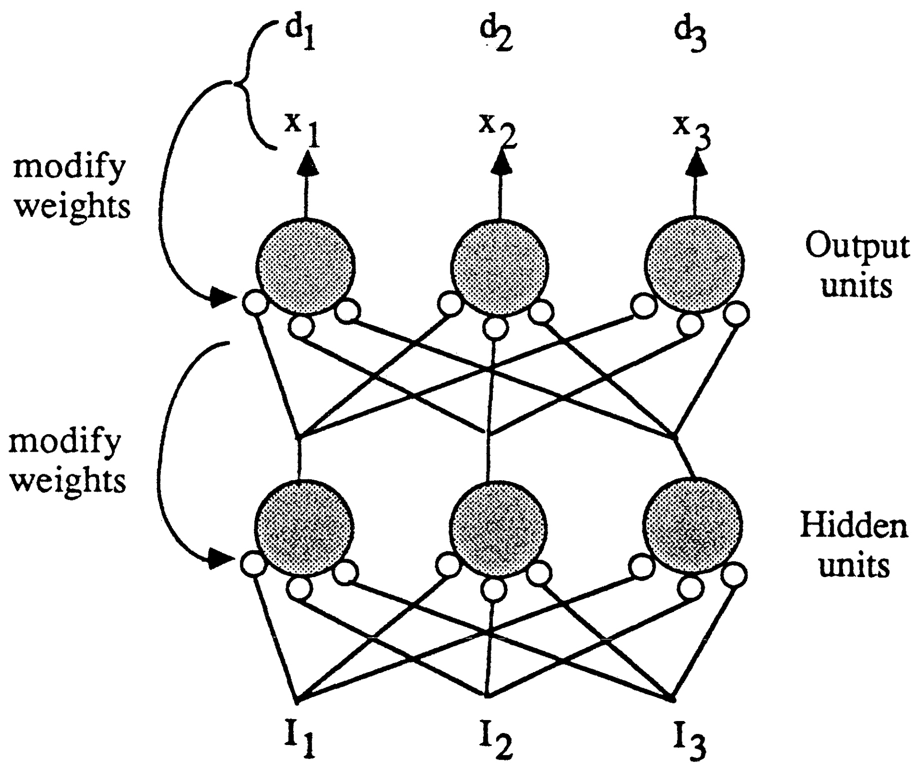


Figure 5.1 Functional Components of Backpropagation Model

tion is the sum of the squares of the errors in the data-transfer values of the units in the output layer. The E in the following equation can stand for either error, to be consistent with other least-squares algorithms, or energy, to be consistent with other Liapunov functions.

$$E(t) = \sum_j (d_{sj} - x_{oj}(t))^2$$

where $x_{oj}(t)$ is a unit in the output layer and d_{sj} is the desired response of the unit as determined externally. Feedback in the model occurs in the propagation of the errors from the output layer to the hidden layers through changes in the weights, as opposed to feedback from the data-transfer values of the other units in the outer-product model. Changes in the model can only occur in response to the errors, so once the overall error has stabilized, the model will be stable.

Despite Rumelhart and McClelland's analysis based on continuous time, the assumption that the backpropagation model is processing continuously is not valid. The weights for each layer of units are changed in a separate processing step. For most applications which use this model, more than one external input vector will be used, so that generalization can be performed by the model. Introduction of a new external input vector to the system causes another discontinuity in processing. The use of continuous analysis requires assumptions about constant data-transfer values and weights. The analysis presented here shows that stability can still be shown using discrete analysis. The advantage of discrete analysis is that assumptions about constant data-transfer values and weights are clarified for each step.

Adaptation for an external input / desired output vector pair, $(\underline{I}_s, \underline{d}_s)$, for the backpropagation model is segmented into a different step for adaptation of the weights for each layer of the model. The first step is to adapt the weights on the output units. In subsequent steps, the weights on the units in each hidden layer are adapted. These steps are repeated for the set of external input vectors and associated desired vectors. For the system to be stable, the adaptation needs to reduce the value of the Liapunov function so that condition 1.5c is satisfied. Ideally, each time the external input /desired output vector pair is changed, the value of the Liapunov function of the model is either unchanged or reduced. In general, this does not happen, since changing the external input vector and the desired vector usually increases the value of the Liapunov function. The system will not converge for a given set of inputs unless eventually, reduction of the Liapunov function due to adaptation exceeds the increase in the value of the Liapunov function caused by changing the external input values.

The first step is the adaptation of the weights of the output units. For this step the output values of the hidden units are assumed to be constant. The change in the value of the Liapunov function at time, t+1, after the weights of the hidden layer are changed from their value at time, t, is:

$$E(t+1) - E(t) = \Delta E = \frac{1}{2} \sum_j [(d_{sj} - x_{oj}(t+1))^2 - (d_{sj} - x_{oj}(t))^2]$$

where d_{sj} is the desired output value of the j^{th} output unit, x_{oj} when the external input vector is $\underline{I}_s$. This equation can be reduced assuming d_{sj} is constant for the time required to change the values of the weights:

$$\Delta E = \frac{1}{2} \sum_j \Delta x_{oj} \; [x_{oj}(t+1) + x_{oj}(t) - 2d_j]$$

where

$$\Delta x_{oj} = x_{oj}(t+1) - x_{oj}(t)$$

Since the Liapunov function is always positive, stability can be shown if ΔE is always non-positive and zero at the equilibrium point. This condition is satisfied if the following conditions are satisfied for each unit in the output layer of the backpropagation model.

$$\Delta x_{oj} = 0, \mathit{or}$$

$$[d_{sj} - x_{oj}(t+1)] + [d_{sj} - x_{oj}(t)] > 0, \text{ and } \Delta x_{oj} > 0, \mathit{or}$$

$$[d_{sj} - x_{oj}(t+1)] + [d_{sj} - x_{oj}(t)] < 0, \text{ and } \Delta x_{oj} < 0.$$

The adaptation process produces these conditions under the assumptions that firstly, the output values of the units are positive and bounded by the values of the desired output vector, and secondly, the threshold function is a monotonically increasing function of the sum of the weighted inputs. The first assumption can be met by restricting the upper and lower bounds of the thresholding function to the values of the desired output vector.

$$d_{sj} \ \varepsilon \left\{ MAX, MIN \right\}$$

This assumption is necessary to ensure that increasing or decreasing the output value of $x_{oj}(t)$ due to weight adaptation does not change the sign of the quantity $d_{sj} - x_{oj}(t)$. The second assumption ensures that $x_{oj}(t)$ increases when $wo_{ij}(t)$ increases for all i. These assumptions translate the above conditions of stability to:

$$\Delta wo_{ij} = 0, \mathit{when} \ d_{sj} - x_{oj}(t) = 0,$$

$$\Delta wo_{ij} > 0, \, when \;\; d_{sj} - x_{oj}(t) > 0, \text{ and}$$

$$\Delta wo_{ij} < 0, \, when \;\; d_{sj} - x_{oj}(t) < 0$$

for all units, j, in the output layer.

The learning rules for the backpropagation models which use the linear thresholding function and the sigmoidal thresholding function both satisfy these conditions. The learning rule for the linear thresholding function is:

$$\Delta wo_{ij} = \eta \; [d_{sj} - x_{oj}(t)] \; x_{hi}(t)$$

The learning rule for the sigmoidal thresholding function is:

$$\Delta wo_{ij} = \eta \, f'(\sum_i w_{o,ij} \; x_{hi}(t)) \; [d_{sj} - x_{oj}(t)] \; x_{hi}(t)$$

where η is a positive constant governing the learning rate, $x_{hi}(t)$ is the data-transfer value of the hidden layer unit connected through weight, wo_{ij}, and $f'(\sum_i wo_{ij} \; x_{hi}(t))$ is the derivative of the threshold function with respect to the sum of the weighted inputs. All of these quantities are defined to be positive. The factor, $d_{sj} - x_{oj}(t)$, determines the sign of the change in weight since all other factors are always positive. Thus, the conditions are satisfied and this step always produces a decrease in the value of the Liapunov function at time, t.

The next steps involve changing the weights of the units in each hidden layer; starting with the layer closest to the output layer and ending with the layer which receives inputs directly from the external input vector. The adaptation function ensures that these steps decrease the value of the Liapunov function according to condition 1.5c. The change in the data-transfer values of the output units caused by the adaptation of the weights for the hidden units has the same sign as the change caused by the adaptation of the weights for the output units. Only the sign of the modification of the output data-transfer values is important, since the data-transfer values are bounded. The magnitude of the modification effects the model's ability to learn responses to other external input vectors, and is controlled by the value of the constant, η. The resulting learning rule for the adaptation of the weights in layer, l, cause the data-transfer values of the units on the layer above, $l-1$, to continue changing in the same direction.

$$\sum_i w_{l-1,ij(t+l)} \Delta x_{li} = 0, \; when \; \Delta x_{l-1,j} = 0,$$

$$\sum_i w_{l-1,ij(t+l)} \Delta x_{li} > 0, \; when \; \Delta x_{l-1,j} > 0, \text{ and}$$

$$\sum_i w_{l-1,ij(t+l)} \Delta x_{li} < 0, \; when \; \Delta x_{l-1,j} < 0.$$

Since the inputs from the lower layer are considered constant and the threshold function is monotonically increasing, the sign of Δx_{li} will be the same as the sign of $\Delta w_{l+1,ki}$. The conditions above can be transformed to suggest an adaptation process:

$$\Delta x_{li} = 0, \; when \; \sum_j w_{l-1,ij(t+l)} \Delta x_{l-1,j} = 0,$$

$$\Delta x_{li} > 0, \; when \; \sum_j w_{l-1,ij(t+l)} \Delta x_{l-1,j} > 0, \text{ and}$$

$$\Delta x_{li} < 0, \; when \; \sum_j w_{l-1,ij(t+l)} \Delta x_{l-1,j} < 0.$$

The value, $\Delta x_{l-1,j}$ can be defined recursively in the manner described in the generalized backpropagation learning rule (with $f' = 1$ for linear thresholding functions):

$$\delta_{l-1,j} = f'(\sum_k w_{l+1,ki}\ x_{l+1,k})\ \Delta x_{l-1,j}$$

$$\delta_{lj} = f'(\sum_k w_{l+1,ki}\ x_{l+1,k})\ \Delta x_{lj}$$

The learning rule is:

$$\Delta w_{l,ik} = \eta\ \delta_{li}\ x_{l+1,k}$$

which satisfies the above conditions since η and $x_{l+1,k}$ are both positive. Therefore, the adaptation of weights in the hidden layers reduces the value of the Liapunov function for adaptation of the model.

The adaptation continues after the external input vector and the associated desired output vector are changed to $(\underline{I}_{s+1}, \underline{d}_{s+1})$. The change in the vector pair causes the Liapunov function to change in value. The value of the desired output vector unit, d_j, changes to $d_{s+1,j}$, so the change in Liapunov function has the form:

$$\Delta E = \sum_j [(d_{s+1,j} - x_{oj}(t))^2 - (d_{sj} - x_{oj}(t))^2]$$

The Liapunov function will increase if the previous output vector is closer in Hamming distance to its desired output vector than the new output vector is to its desired output vector. The adaptation process which took place for the previous vector usually ensures that the Liapunov function will increase at this step. Therefore, stability cannot be ensured in general. None-the-less, for specific applications, weights which yield a zero value of the Liapunov function exist and are usually obtained by the adaptation process. If the model avoids local minima and excessive oscillation, then the backpropagation learning algorithm will adapt to an optimal set of weights.

If the pairs of vectors to be learned by the model are orthogonal, then a set of weights exist for which the model is stable of the form:

$$W = \sum_s \underline{d}_s \cdot \underline{I}_s^T$$

For these effective weights between the external input vector, $\underline{I}$, and the activation levels of the output units,

$$u_{oj} = \sum_s \underline{d}_s \cdot \underline{I}_s^T \cdot \underline{I}$$

If $\underline{I}$ is one of the learned, orthogonal inputs, then

$$u_{oj} = |\underline{I}|^2 \underline{d}$$

As in the outer-product model, the constant, $|\underline{I}|^2$ is eliminated with the threshold function. This results in $\underline{x}_o = \underline{d}$ which results in the Liapunov function having the value zero. Fortunately, Kolmogorov has shown that any continuous function of N variables has an optimal set of weights in a two-hidden layer backpropagation type model with N(2N+1) units[22] . Therefore, a global minimum in the Liapunov function does exist for most applications, but this minimum will not be accessed by the model if the processing is caught in a local minimum.

One approach to avoiding the local minima is the addition of a momentum term in the adaptation function:

$$\Delta w_{ji}(t) = \eta\ \delta_j\ i_i + \alpha\, \Delta w_{ji}(t-1)$$

This term causes Δw_{ji} to go to zero gradually so that the weights do not stabilize at local minima. The disadvantage of this modification is that the adaptation steps are no longer guaranteed to reduce the Liapunov function. However, if α is small enough, the increase will not be appreciable. The size of α is dependent upon the application.

Since stability cannot be proved in general, the restrictions on the parameters which result from this analysis are merely guidelines. To be consistent with the least-mean squares algorithm, the desired output vectors are binary, and the thresholding function is bounded by the minimum and maximum of the desired output values (see table 5.1). Any monotonically increasing thresholding function can be used. The parameters η and α are application dependent. Both parameters need to be large enough to reduce the Liapunov function and avoid local minima, but small enough so that the model converges in a reasonable amount of time. The simulation analysis in the next chapter gives examples of the effects of the sizes of the parameters. The simulations also illustrate the backpropagation's ability to find the global minimum for the Liapunov function when local minima are present.

Table 5.1 Restrictions on Backpropagation Model

Connections	Weights and Adaptation	Decision Function
1) Binary desired output values 2) Multiple layers of units 3) Only feed-forward connections between units of adjacent layers	1) Adaptation is proportional to change in energy	1) Threshold function is bounded by minimum and maximum values of desired output values 2) Transfer function is monotone increasing 3) Activation level is: $\sum_j w_{ji} x_j$

Chapter 6
EXAMPLE OF PARAMETER ANALYSIS

6.1 Choice of Model and Parameters

The mathematical analysis performed in the four preceding chapters is appropriate for understanding the general limitations of the models, but complete mathematical analysis of specific applications is not practical. Each parameter needs to be optimized with respect to the other parameters, and the external input vectors need to be shown to be within the scope of a stable model. Another way of performing this in depth analysis is through simulation of the application. Simulation has the advantages of being more intuitively appealing, and allowing tests for situations which were simplified in the mathematical analysis. The disadvantages of simulations are that the results are difficult to generalize, and the simulation may falsely predict or mask problems in the actual implementation due to approximations.

The simulations performed here demonstrate the usefulness of the Liapunov function as a measure of the performance of a model, even when the conditions of the second method of Liapunov are not satisfied. This method of stability analysis based on simulation is particularly useful for models such as the backpropagation model and adaptive resonance theory which do not satisfy all the conditions of Liapunov's second method in general.

An example of stability analysis using a software simulation is presented in this chapter. By plotting the Liapunov function versus time, visual inspection can determine if the Liapunov function appears to be asymptotically approaching its limit, even if the function does not strictly satisfy the conditions of Liapunov's second method for all time. If oscillations persist or if the Liapunov function stabilizes at a point other than at its limit, the model has probably gone into a limit cycle or local minimum. Limit cycles and local minima are difficult to predict analytically for more than two units, but the characteristic oscillations and inappropriate stabilization in the Liapunov functions are easy to identify from simulation. Liapunov functions with very slow convergence times are also readily identified with simulations. Limit cycles, local minima and slow convergence are the primary problems with dynamical systems, and are the conditions to be checked for a stable neural-network model application.

The backpropagation model is used in this presentation because of its simplicity. Adaptive resonance theory is more complex and has a relatively large number of parameters so simulation of adaptive resonance theory would be more difficult to interpret than the simulation presented here. The simulation of the backpropagation model followed the description of the generalized delta rule (the backpropagation adaptation process) presented in Rumelhart and McClelland's book, *Parallel Distributed Processing*. Several versions and implementation details are presented in this book. The choice of version and implementation details for this simulation is based on the desire to achieve a model with interesting dynamics, yet simple functional components.

A single hidden layer connection scheme was chosen with input units represented by the external input vector. Multiple hidden layers may provide better internal representation, but for the small model simulated here, the additional layers would probably not change the behavior of the Liapunov function enough to justify the additional complexity. The number of units, four per layer, is small, but the network is large enough to demonstrate some interesting dynamics.

The adaptation of the weights is in a form which closely approximates continuous processing. For each external input/desired output pair in the set of patterns to be learned, a pair is presented to the model, an error is generated, and the weights of the output units and the hidden units are adapted simultaneously. This procedure is repeated for a number of iterations to provide insight into the long term behavior of the Liapunov function. An alternative method of adaptation is to calculate the weight changes after the presentation of each pair, but wait to change the values of the weights until all of the pairs of vectors in the set have been presented. This second method of adaptation is claimed to be more stable, but the process is a much weaker approximation to a continuous system. The weaker approximation may cause the Liapunov function to mask some instability. The first method is therefore chosen for this analysis.

The equations for adaptation are the same as the equations including the momentum term, presented in chapter 5.

$$\Delta w_{ji}(t) = \eta\, \delta_j(t)\, I_i + \alpha\, \Delta w_{ji}(t-1)$$

The form of $\delta_j(t)$ is simple for both weight matrices due to the the simplicity of the derivative of the threshold function. For the output layer $\delta_{oj}(t)$ is:

$$\delta_{oj}(t) = x_{oj}(t)\,(1 - x_{oj}(t))\,(d_j - x_{oj}(t)) \tag{6.1}$$

and for the hidden units, $\delta_{hj}(t)$ is:

$$\delta_{hj}(t) = x_{hj}(t)\,(1 - x_{hj}(t)) \sum_j wo_{ji}(t)\, x_{oj} \tag{6.2}$$

The decision function chosen is the most commonly used form for a decision function: the activation level is the sum of the weighted inputs from all of the units in the previous layer, and the threshold function is sigmoidal.

$$x_i(t) = \frac{1}{1 + e^{-u_i(t)}} = \frac{1}{1 + e^{-\sum_j w_{ji}(t)\, x_j(t)}}$$

The upper and lower bounds for the sigmoidal functions are one and zero, respectively, which corresponds to the values of the elements of the desired output vector. The center of the sigmoidal function, T_i, is at zero activation level for symmetry between negative and positive activation levels, which simplifies analysis. The derivative of the sigmoid threshold function is used in the equations 6.1 and 6.2 for δ. The derivative is:

$$\frac{e^{-u_i}}{(1 + e^{-u_i})^2} = x_i\,(1 - x_i)$$

The experiments performed show differences in the Liapunov function for different sets of external input / desired output pairs and values of α and η. The differences in the pairs of vectors are denoted by the orthogonality and the linear independence between successive vectors. The value of the rate of learning parameter, η, and the momentum parameter, α, are varied to show their effect on convergence of the Liapunov function. These experiments are certainly not comprehensive but the experiments do support the conclusions of the mathematical analysis and raise some interesting questions. Future research could include a set of more systematic experiments for a specific application as opposed to the general application presented here.

6.2 Simulation Results

Some simple experiments with the simulation of the backpropagation model are presented here as an example of the usefulness of simulation analysis of stability based on the Liapunov function. Computer software for the simulation is written in the C programming language and presented in Appendix A. The experiments consisted of learning three different external input / desired output vector pairs under variations of the parameters. The results of the experiments show the model's ability to learn orthogonal, linear independent and linearly dependent sequences of vector pairs with appropriate values of the parameters α and η.

6.2.1 Results for Orthogonal Data

The first set of data used has external input vectors which are orthogonal to each other, and desired output vectors which are orthogonal to each other. This set of vector pairs has a known optimal weight matrix. The matrices which result from the adaptation are compared to this known matrix to provide another measure of the performance of the backpropagation model to compare with the Liapunov function.

The vectors used have a Hamming distance of two between successive external input vectors and successive desired output vectors. The Hamming distance did not seem to effect the stability of the Liapunov function in other experiments not presented here, but the Hamming distance may be more of an issue for larger vectors. The external input / desired output vector pairs used in this experiment are:

$$\underline{I}_1 = [1, 0, 0, 0]^T, \quad \underline{d}_1 = [0, 0, 0, 1]^T$$
$$\underline{I}_2 = [0, 1, 0, 0]^T, \quad \underline{d}_2 = [0, 0, 1, 0]^T$$
$$\underline{I}_3 = [0, 0, 1, 0]^T, \quad \underline{d}_3 = [0, 1, 0, 0]^T$$

Other than the constraints mentioned above, the choice of vectors was arbitrary.

The learning rate, η and the momentum, α, are varied between two values each:

$$\eta = 0.1 \text{ or } 0.99$$
$$\alpha = 0.2 \text{ or } 0.9$$

These values are also somewhat arbitrary other than the condition that one of the values is large and the other is small for this application. The experiment consisted of performing a simulation of the model at each of the four combinations of the above values of the parameters η and α. Each pair was presented to the model fifty times to simulate the time behavior of the

Liapunov function.

The results of the simulation showed that the relative size of the parameters η and α is important to the speed of convergence of the Liapunov function and the ability of the model to avoid local minima. When both η and α are small, the convergence is slow and fairly smooth. When either η or α is large, the convergence rate increased and for this set of vectors, the Liapunov function stabilized at a value above zero. This behavior of the model corresponds to the adaptation being caught in a limit cycle or a local minimum. Figure 6.1 shows a plot of the Liapunov function versus the order of the presentation of the vector pairs. The first value plotted is the value of the Liapunov function when the first pair, $(\underline{I}_1, \underline{d}_1)$ is presented to the model, then the next point is the value of the Liapunov function after the weights are adapted. The next two values correspond to the presentation and adaptation of the weights for the second pair, $(\underline{I}_2, \underline{d}_2)$ and the following two values correspond to the presentation and adaptation of $(\underline{I}_3, \underline{d}_3)$. This series of presentations and weight adaptations are repeated fifty times. The plot shows that the error continues to change due to weight adaptation even though the error for the presentation of the vector pairs has stabilized. This result suggest that the Liapunov function is stuck in a local minimum in which the values of the weights can change, but not in the manner which causes them to be closer to their optimal values. This phenomena can also be considered to be a limit cycle in which the weight changes but the state of the backpropagation model stays constant.

When both η and α are large, then the Liapunov function overcomes the local minimum to find the global minimum. Figure 6.2 shows that both the

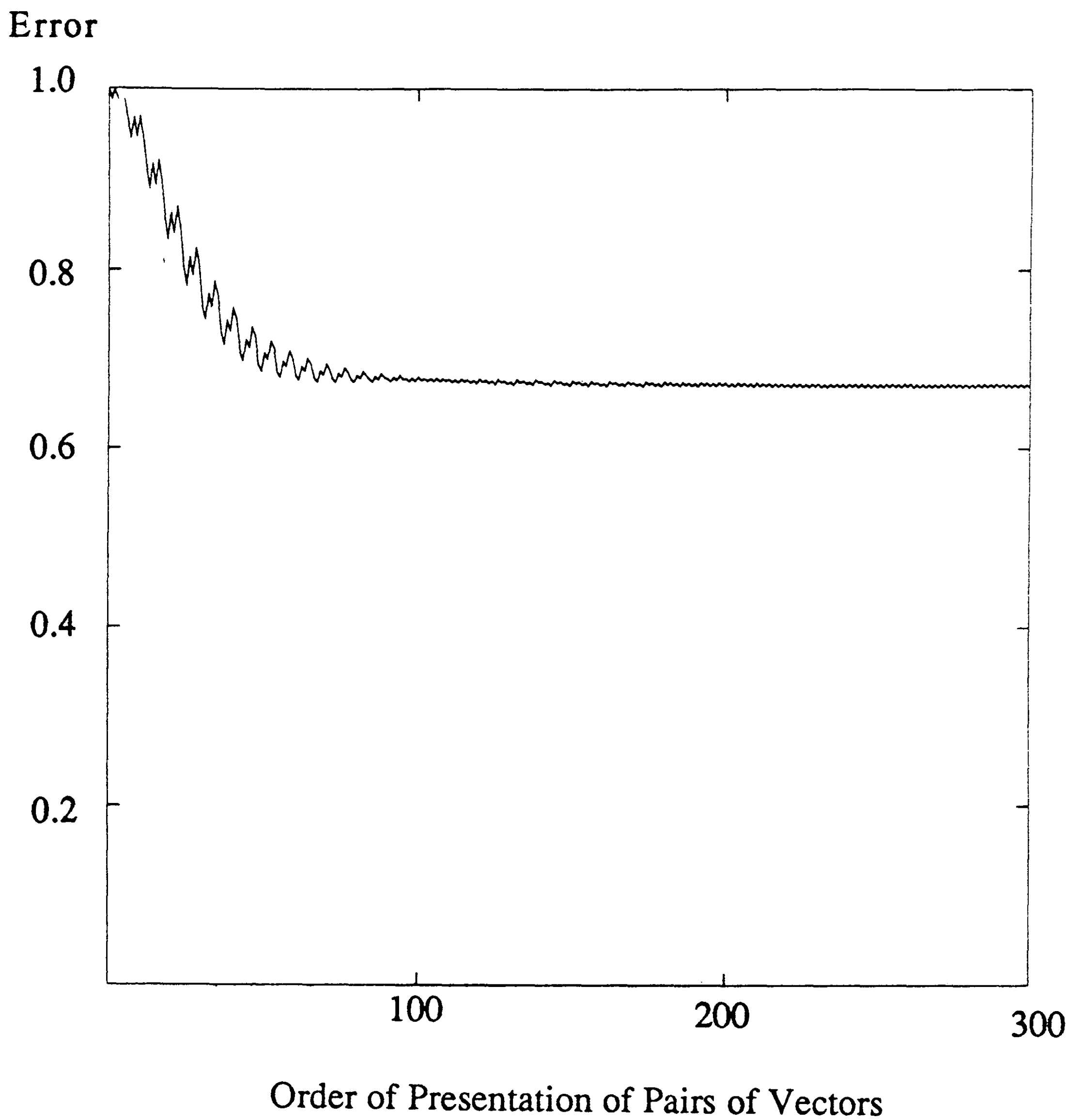


Figure 6.1 Local Minimum in Liapunov Function of Backpropagation Model

value of the Liapunov function and its discrete change appear to be approaching zero. The Liapunov function satisfies the conditions of equation 1.5 of Liapunov's second method as $\underline{x}_o$ approaches $\underline{x}_{oeq}$, even though the function certainly does not satisfy condition 1.5c for early presentations of the vector pairs.

The final values of the weights reinforce the belief that the backpropagation model has found the global minimum of the Liapunov function. As was shown in section 5.2, an optimal set of weights for orthogonal external input vectors is the sum of the outer products of the desired output vectors and the corresponding external input vectors.

$$w = \sum_{s=1}^{3} \underline{d}_s \cdot \underline{I}_s{}^T = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

The adapted weights, wo and wh, should combine to approximate w when the threshold function is considered.

$$wh \cdot wo = \begin{bmatrix} -1.89 & -0.77 & -0.98 & -3.26 \\ 7.09 & -15.72 & 28.74 & 2.19 \\ 3.99 & 30.42 & -14.21 & 3.70 \\ -5.30 & -12.82 & -12.69 & 5.21 \end{bmatrix}$$

with the exception of w_{41}, the weight values below eight correspond to zeros in w, while large positive values correspond to ones. Therefore, the adapted weight matrices are consistent with the theoretical matrix. Thus, the belief is re-enforced that the backpropagation model is successful in learning the three orthogonal vector pairs.

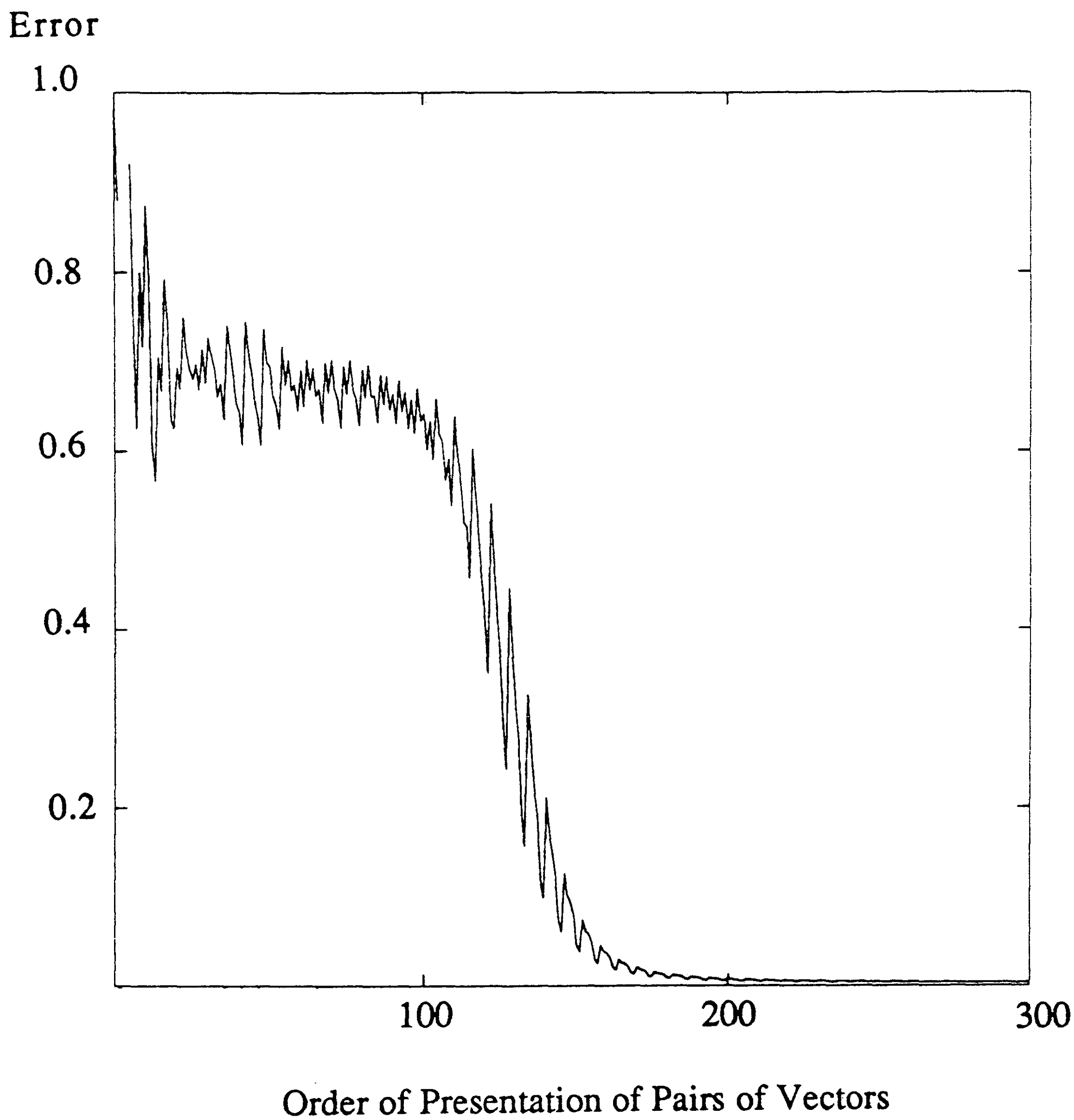


Figure 6.2 Convergence of Backpropagation Model with Orthogonal Data

6.2.2 Results for Linearly Independent Data

The next experiment used non-orthogonal, but linearly independent external input / desired output vector pairs. The same four combinations of values for η and α were used in the simulations. If the density of the vectors is more important than the orthogonality of the vectors then this set of vector pairs which have more elements with the value one should have converged faster than the sparse vector pairs of the first experiment. This result was not observed.

The vectors used have the same Hamming distance between the successive vectors as in the first experiment. The vector pairs are:

$$\underline{I}_1 = [1, 0, 1, 1]^T, \quad \underline{d}_1 = [0, 1, 1, 1]^T$$
$$\underline{I}_2 = [1, 1, 0, 1]^T, \quad \underline{d}_2 = [1, 1, 1, 0]^T$$
$$\underline{I}_3 = [1, 0, 0, 0]^T, \quad \underline{d}_3 = [0, 1, 0, 0]^T$$

The values for α and η are the same as for the first experiment.

For small values of α and / or η the Liapunov function stabilizes at a local minimum in a manner similar to the same situation in the first experiment. For large values for both η and α, the Liapunov function again was able to escape the local minimum and find the global minimum as seen in figure 6.3. The Liapunov function and its discrete change approach zero so that the conditions of Liapunov's second method as listed in equation 1.5 are satisfied for later times, even though the early behavior of the Liapunov function is clearly oscillatory.

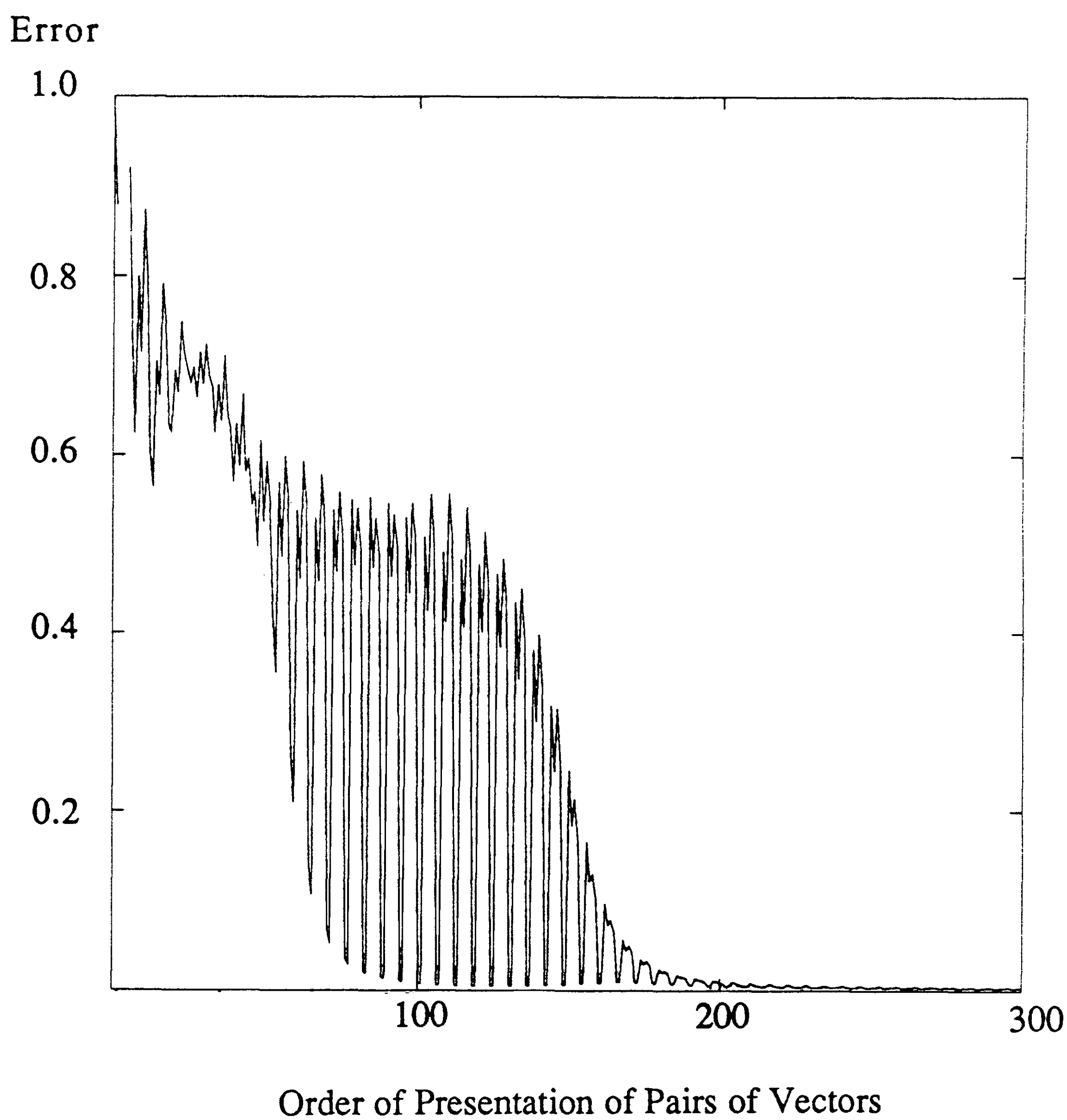


Figure 6.3 Convergence of Backpropagation Model with Linearly Independent Data

Despite the larger weight changes before equilibrium is approached, the non-orthogonality of the vector pairs appeared to slow the convergence rate. The difference in convergence rate as compared to experiment one is not extremely significant (10-20 presentations of the vector pairs). The change in convergence rate may be caused by other factors such as the large oscillations in the Liapunov function.

6.2.3 Results for Linearly Dependent Data

The last experiment used vectors with successive input vectors which are linearly dependent. The ability of the backpropagation model to adapt to linearly dependent vectors is one of the model's most interesting capabilities.

The same parameter variations were performed for the linearly dependent vectors as in the other two experiments. The external input / desired output vector pairs for this experiment are:

$$\underline{I}_1 = [1, 0, 0, 0]^T, \quad \underline{d}_1 = [0, 0, 0, 1]^T$$
$$\underline{I}_2 = [0, 1, 0, 0]^T, \quad \underline{d}_2 = [0, 0, 1, 0]^T$$
$$\underline{I}_3 = [1, 1, 0, 0]^T, \quad \underline{d}_3 = [0, 1, 0, 0]^T$$

These vectors are sparse and closely resemble the vector pairs used in the first experiment.

The results of the last experiment are very similar to the results for the other experiments. When η and / or α is small, the model finds a local minimum. When both parameters are large, the model proceeds out of the local minimum towards a stable global minimum (see figure 6.4). The convergence rate is the slowest of the three experiments. The speed of convergence is possibly due to the complexity of determining the values of the weights for vector pairs which are not orthogonal or linearly independent. No obvious restriction exists for the external input / desired output pairs to avoid the model stabilizing at local minima.

6.2.4 Overall Results

An interesting result of these experiments is the apparent need for both η and α to be large to overcome local minima. The best size of these parameters depends on the "depth" of the local minima for specific application. Simulation is a good way to search for the values of the parameters since local minima can usually be easily identified visually.

The global minimum is also easily identifiable in that when the model is approaching the global minimum, the Liapunov function satisfies the conditions of Liapunov's second method. The simulation approach allows the oscillations in earlier periods of the Liapunov function to be removed from consideration in a straight-forward manner which would not be possible using a strictly mathematical approach.

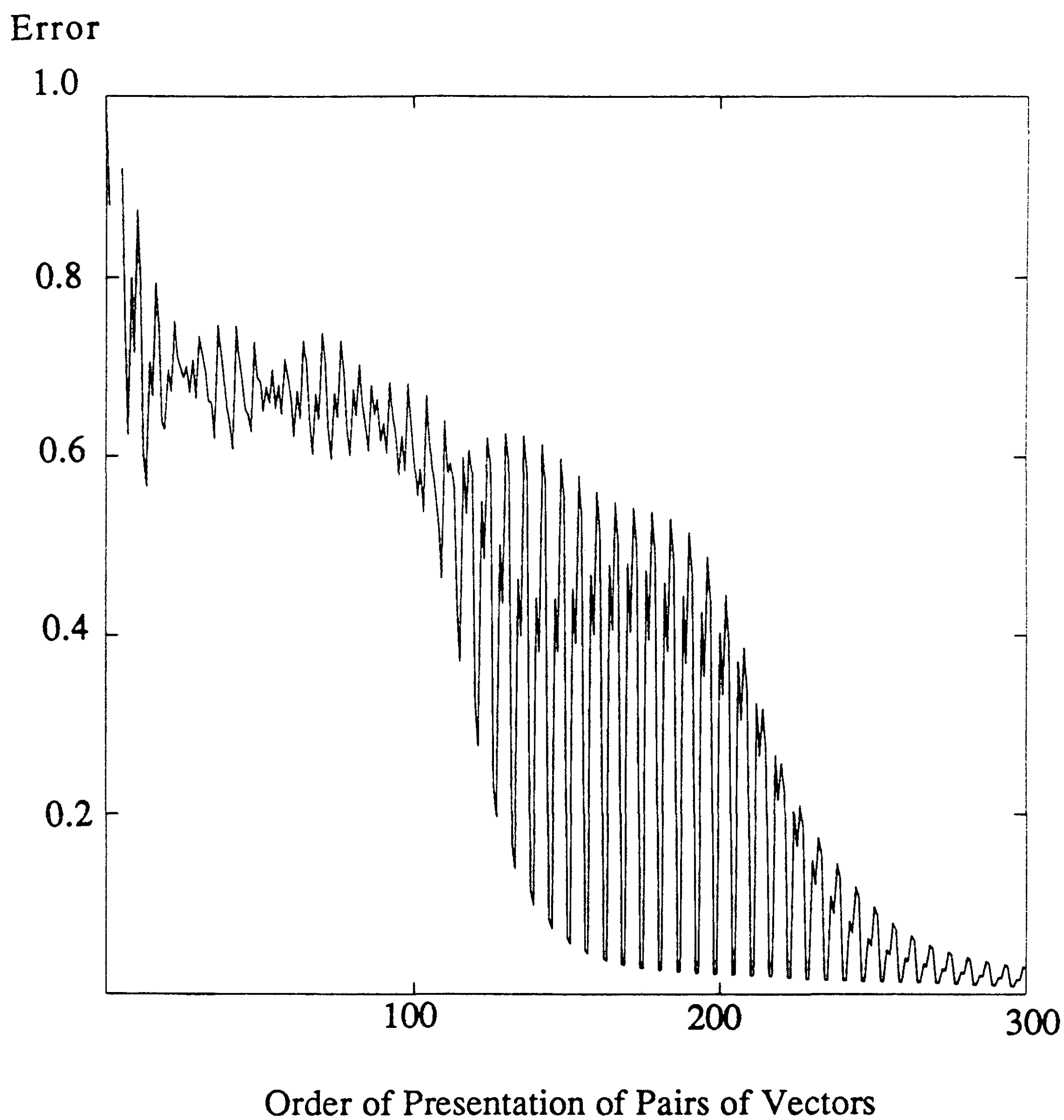


Figure 6.4 Convergence of Backpropagation Model with Linearly Dependent Data

Chapter 7
CONCLUSIONS

The primary difference between models is in how dynamics are implemented (see table 7.1). The outer-product model uses processing dynamics; the backpropagation model uses adaptation dynamics; and the Boltzman machine and adaptive resonance theory use a combination of the two types of dynamics. With the exception of adaptive resonance theory's phases 1, 3, 4 and 5, the dynamics of the models depend on the interaction of the units which can be seen in the terms of the Liapunov functions. This interaction between units is on of neural-network models' salient features. The interaction provides interesting processing dynamics. The complexity of these dynamics results in the need for Liapunov functions to determine if the model is performing in a stable manner.

The form of the Liapunov function can be seen to correspond to well known algorithms in other fields. The Liapunov functions for the outer-product model, the Boltzman machine adaptation and the backpropagation model are similar to Liapunov functions from optimization theory: the steepest decent algorithm and the least-squares algorithm. The Liapunov function of adaptive resonance theory is similar to the Liapunov function for the clustering algorithm[23] . This analogy suggests that the outer-product model, the Boltzman machine and the backpropagation model will work well with optimization applications, while adaptive resonance theory will work well with signal processing applications. Successful applications of

Table 7.1 Comparison of Liapunov Functions

	Type of Dynamics	Liapunov Function
Outer-product Model	Steepest descent algorithm	Processing dynamics $-\frac{1}{2}\sum\sum_{i\neq j} w_{ij}x_j x_i - \sum_i I_i x_i + \sum_i T_i x_i$ (discrete model) Processing dynamics $-\frac{1}{2}\sum\sum_{i\neq j} w_{ij}x_j x_i - \sum_i I_i x_i + \frac{1}{R_i}\int_0^{x_i} f_i(x)dx$ (continuous model)
Boltzman Machine	Simulated anealling algorithm	Processing dynamics $-\frac{1}{2}\sum\sum_{i\neq j} w_{ij}x_i x_j$ Adaptation dynamics $\sum_s P(I_s, d_s) \ln \frac{P'(x_{ii}, x_{oj})}{P(I_s, d_s)}$
Backpropagation Model	Least squares algorithm	Adaptation dynamics $\sum_i (d_{si} - x_i)^2$
Adaptive Resonance Theory	Clustering algorithm	Processing dynamics $\sum_i \left\| u_{1i} - \frac{1 + D_1 \sum_j wt_{ji}x_{2j} + B_1 \sum_j x_{2j}}{1 + A_1 I_i + A_1 D_1 \sum_j w_{ji} x_{2j} + C_1 \sum_j x_{2j}} \right\|$ Phase 1 and 3 (F1) Processing dynamics $\max\left(\frac{u_j}{\sum_k u_k}\right) - \min\left(\frac{u_j}{\sum_k u_k}\right)$ Phase 2 (F2) Adaptation dynamics $\sum_i \sum_j wt_{ij}$ Phase 4 (F1) Adaptation dynamics $\sum_i \sum_j wb_{ij}$ Phase 5 (F2)

the models have tended to fall along these lines. The optimization problem referred to as the traveling salesman problem of finding the shortest route between a group of cities was successfully processed by the outer-product model in the original papers. In NETalk[24] , Sejnowski, et al used the backpropagation model and the Boltzman machine to optimize the connections between a set of letters in text and a set of phonemes. Adaptive resonance theory has been applied to signal processing problems in vision and control theory[25] . A choice of a neural-network model for future applications can be simplified if these guidelines are used. In addition, many of the lessons learned in the fields of optimization theory and signal processing can be applied to the analogous neural-network models.

The stability analysis of the outer-product model shows the differences in the analyses for continuous and discrete processing dynamics. Different threshold functions are required in the two versions of the outer-product model to ensure stability. In the discrete version, the asynchronous hard-limiting threshold function is used. This function can be thought of the discrete approximation to the synchronous sigmoidal threshold function used in the continuous version. Since computer simulations are discrete approximations of continuous models, this modification to the decision function should be considered in the computer simulation of other continuous neural-network models.

The analysis of the Boltzman machine shows how both processing and adaptation dynamics are used in a stochastic neural-network model. The processing dynamics are a stochastic extension of the dynamics of the discrete outer-product model in a multi-layer model. This extension is useful if the intermediary representations of the hidden layer aid in the processing of an application, and local minima in the Liapunov function are to be avoided. The Boltzman machine is also an improvement over the outer-product model if the statistical distribution of the external input / desired output vector pairs is appropriate for determining the values of the weights. Statistical inference as is performed in the adaptation of the weights in the Boltzman machine, requires a great deal of information which slows the adaptation process. The stochastic approach to processing dynamics is also slow. Therefore, the Boltzman machine is only appropriate for applications which have no simple solution and can afford to trade processing time for adaptability and processing which avoids local minima.

For most applications in which both adaptation and processing dynamics are required, adaptive resonance theory is more appropriate. The five phases of the model are complex, but each phase can easily be approximated by a solution for the state of the model which does not require dynamical processing. These approximations greatly increase the speed of the model on a serial computer. With or without these approximations, the weights are not necessarily stable. The stability depends upon the the vigilance level for a particular application. When the vigilance level is large, the weights are more likely to be stable, but the inputs are stored in more units. When the vigilance level is low, the inputs are stored in fewer units, so generalization is performed, but the weights are less likely to be stable.

The backpropagation model also requires an understanding of the application to achieve stability since this model is not stable in general. The backpropagation model is very susceptible to getting caught in local minima in the Liapunov function. The momentum term in the adaptation function helps the backpropagation model avoid the local minima, but the term does not guarantee that the global minimum for the Liapunov function will be found. The size of this and other parameters is dependent upon the application.

Simulation of the Liapunov function is a good method for determining parameters to ensure stability for a specific application in models such as adaptive resonance theory and the backpropagation model. Simulation is useful for identifying local minima in the Liapunov function and monitoring the progress of the model relative to the proximity of the Liapunov function to its bounds. Monitoring the progress, or convergence rate of the model is useful for models which are known to be stable, such as the outer-product model and the Boltzman machine, as well as for the conditionally stable models.

The analyses in chapters 2 through 6 show that Liapunov functions are useful for understanding the dynamics of neural-network models, regardless of whether the model has absolute asymptotic stability. For the models which do possess stability according to Liapunov's second method, such as the outer-product model and the Boltzman machine, restrictions on parameters are clarified. For the models which cannot be shown to be stable in general, such as adaptive resonance theory and the backpropagation model, the Liapunov function is useful for pointing out potential sources of instability and suggesting parameter restrictions. Simulation is an approach for determining stability for specific applications of the models which are not stable in general.

REFERENCES

1. A.F. Murray and A.V.W. Smith, "Asynchronous VLSI Neural Networks using pulse-stream arithmetic." *IEEE Journal of Solid State Circuits* 23, no.4 (1988): 688-697.

2. D.E. Rumelhart and J.L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Cambridge, MA: MIT Press, 1987.

3. F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological Review* 65 (1958): 386-408.

4. N. Wiener, *Cybernetics.* New York: MIT Press, 1961.

5. M. Minsky and S. Papert, *Perceptrons.* Cambridge, MA: MIT Press, 1969.

6. J.J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities." *Proceedings of the National Academy of Science, USA* 79 (1982): 2554-2558.

7. A.M. Liapunov, *Stability of Motion.* New York: Academic Press, 1966.

8. D. Hestenes, "How the brain works: the next great scientific revolution." In *Maximum Entropy and Bayesian Spectral Analysis and Estimation Problems* Ed. C. Ray Smith, Boston: Reidel, Dordecht, 1986.

9. J.J. Hopfield and D.W. Tank, "'Neural' computation of decisions in optimization problems." *Biological Cybernetics* 52 (1985): 141-152.

10. D.G. Bounds, "New optimization methods from physics and biology."

Nature 329, no. 17 (1987): 215-219.

11. A.N. Michel, J.A. Farrell and W. Porod, "Stability Results for Neural Networks." In *Neural Information Processing Systems,* Ed. D.Z. Anderson, 554-563. American Institute of Physics, New York, 1988.

12. S. Geman and C. Graffigne, "Markov random field image models and their applications to computer vision." In *Proceedings of the International Congress* of Mathematicians. 1496-1517. Berkeley, CA, 1986.

13. N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equation of state calculations for fast computing machines." *Journal of Chemical Physics.* 6 (1953): 1087.

14. S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi, "Optimization by simulated annealing." *Science.* 220 (1983): 671-680.

15. D.H. Ackley, G.E. Hinton, and T.J. Sejnowski, "A learning algorithm for Boltzman machines." *Cognitive Science.* 9 (1985): 147-169.

16. S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and Bayesian restoration of images." *IEEE Transactions on Pattern Analysis and Machine Intelligence.* PAMI-6 (1984): 721-741.

17. S. Grossberg (ed.), *Neural Networks and Natural Intelligence.* Cambridge, MA: MIT Press, 1988.

18. G.A. Carpenter and S. Grossberg, "A massively parallel architecture for a self-organizing neural pattern recognition machine". *Computer Vision, Graphics, and Image Processing.* 37 (1987): 54-115.

19. S.A. Ellias and S. Grossberg, "Pattern formation, contrast control, and oscillation in the short term memory of shunting on-center off-

surround networks." *Biological Cybernetics*. 20 (1975): 69-98

20. S. Grossberg, "Contour enhancement, short term memory and constancies in reverberating neural networks." *Studies in Applied Mathematics*. LII, no. 3 (1973): 213-257.

21. D.E. Rumelhart, G.E. Hinton an R.J. Williams, "Learning Internal Representation by Error Propagation." *Parallel Distributed Processing*. Cambridge, MA: MIT Press, 1987.

22. R.P. Lippman, "An Introduction to Computing with Neural Networks." *IEEE ASSP Magazine* 4, no. 4 (1987): 4-22.

23. J. Makhoul, S. Roucos, and H.Gish, "Vector Quantization in Speech Coding." *IEEE Proceedings*. 73 (Nov. 1985): 1551-1588.

24. T.J. Sejnowski and C.R. Rosenberg, "NETtalk; A Parallel Network that Learns to Read Aloud." *Johns Hopkins Univ. Technical Report* JHU/EECS-86/01 (1986).

25. D. Bullock and S. Grossberg, "A neural network architecture for automatic trajectory formation and coordination of multiple effectors during variable-speed arm movements," In *Proceedings of the IEEE First Annual International Conference on Neural* Networks. San Diego (1987).

Appendix A

Software Simualtion of Backpropagation Model

```
  /* backprop2.c */
/* This program accepts a series of input vectors and
associated desired output vectors from files and computes the
adaptation of the weights and the energy in a two layer
backpropagation neural network. Input files, a weight matrix
and an output file are named when calling the program. */

#include <stdio.h>
#include <math.h>

#define NUMUNITS 4      /* number of units in each layer */
#define NUMINPUTS 500   /* max. number of different input, */
            /* desired output pairs */
#define ITERATIONS 50   /* number of presentations of input, */
            /* desired output pairs */
#define ETA .99         /* adaptation rate */
#define ALPHA 0.90      /* momentum constant */

main(argc,argv)
int argc;           /* number of input files + output file */
char *argv[];       /* name of input files and output file */
{ float i[NUMUNITS];    /* input vector */
  float d[NUMUNITS];    /* desired output vector */
  float h[NUMUNITS];    /* data transfer values of hidden units */
  float o[NUMUNITS];    /* data transfer values of output units */
  float delta[NUMUNITS]; /* backpropagated error */
  float energy[NUMINPUTS];      /* energy of the system */
  float wh[NUMUNITS][NUMUNITS]; /* weights between the inputs and */
                 /* the hidden units */
  float wo[NUMUNITS][NUMUNITS]; /* weights between hidden units and */
                 /* output units */
  float dwh[NUMUNITS][NUMUNITS]; /* previous weight changes */
                 /* between the inputs and */
                 /* the hidden units */
  float dwo[NUMUNITS][NUMUNITS]; /* previous weight changes */
                 /* between hidden units and */
                 /* output units */
  float normo,normh;   /* norms of the weight matrices */
  int fno;          /* counter for presentation of inputs */
             /* (file number) */
  int k,j;          /* counters for units */
  int l;            /* counter for iterations */
  double floor();     /* math function, rounds down */
  double sqrt();      /* math function */

  FILE *in;
  FILE *out;          /* file pointers */
```

```
memset(dwh,0x00,NUMUNITS*NUMUNITS*sizeof(float)); /* initialize */
memset(dwo,0x00,NUMUNITS*NUMUNITS*sizeof(float)); /* weight changes */
if(in=fopen(argv[argc-2],"rt"))  /* get weight matrices */
  {
  k=0;
  while(fscanf(in,"%f/n",&wo[(int)floor(k/NUMUNITS)]
[(int)(k-floor(k/NUMUNITS)*NUMUNITS)]) !=EOF && k<NUMUNITS*NUMUNITS)k++;
  k=k-1;
  while(fscanf(in,"%f/n",&wh[(int)(floor(k/NUMUNITS)-NUMUNITS)]
[(int)(k-floor(k/NUMUNITS)*NUMUNITS)])!=EOF &&
k<2*NUMUNITS*NUMUNITS)k++;
  fclose(in);
        }
else
  {
  memset(wh,0x00,NUMUNITS*NUMUNITS*sizeof(float));  /* initialize */
  memset(wo,0x00,NUMUNITS*NUMUNITS*sizeof(float));  /* weights   */
        }

if ((out=fopen(argv[argc-1], "wt"))==0x00)  /* open output file */
  {
  printf("can't open output file!! %s", argv[argc-1]);
  exit(0);
  }

for(l=0;l<ITERATIONS;l++)  /* repeat the presentation of the sets */
  {
  for(fno=1;fno<NUMINPUTS && fno<argc-3;fno+=2)  /* compute energy for */
                                  /* each input file */
    {
    if(in=fopen(argv[fno],"rt"))  /* get input vector */
      {
      k=0;
      while(fscanf(in,"%f/n",&i[k])!=EOF && k<NUMUNITS)k++;
      fclose(in);

      if(k==NUMUNITS)  /* the size of the input vector is correct */
               /* get desired vector */
        {

        if(in=fopen(argv[fno+1],"rt"))
          {
          j=0;
          while(fscanf(in,"%f/n",&d[j])!=EOF && j<NUMUNITS)j++;
          fclose(in);

          if(j==NUMUNITS)  /* the size of the desired vector is correct */
```

```
      {/*compute data transfer values, weights and energy */
      output(i,wh,h);     /* output of hidden units */
      output(h,wo,o);     /* output of output units */
      error(o,d,&energy[(argc-3)*l+fno-1],delta);/*compute error & energy*/
      weight(delta,wo,dwo,h);     /* compute weights for output units */
      weight(delta,wh,dwh,i);     /* compute weights for hidden units */
      output(i,wh,h);     /* recompute output of hidden units */
      output(h,wo,o);     /* recompute output of output units */
      error(o,d,&energy[(argc-3)*l+fno],delta); /*compute error & energy*/

      printf("\nThe energy for the %dth",(argc-3)*l+fno-1);
                    printf(" and the %dth computation",(argc-3)*l+fno);
      printf(" is %f",energy[(argc-3)*l+fno-1]);
      printf(" and %f",energy[(argc-3)*l+fno]);
      }

     else /*wrong size of desired output vectors*/
       {
       printf("\nWrong number of units (%d) in file %s", k, argv[fno+1]);
       exit(0);
       }

     }

     else /* couldn't open file */
      {
      printf("\nCouldn't find %s!!",argv[fno+1]);
      exit(0);
      }

    }

   else /*wrong size of input vectors*/
    {
    printf("\nWrong number of units (%d) in file %s", k, argv[fno]);
    exit(0);
    }

   }

  else /* couldn't open file */
   {
   printf("\nCouldn't find %s!!",argv[fno]);
   exit(0);
   }

}
```

```
 }

 for(l=0;l<(argc-3) * ITERATIONS*2;l++)  /* send energy values to output file */
  {
  fprintf(out,"%f\n",energy[l]);
  }
 fclose(out);

if ((out=fopen(argv[argc-2], "wt"))==0x00)   /* open output file */

 {
 printf("can't open output file!! %s", argv[argc-1]);
 exit(0);
 }
else
 {
 normo = 0.00;
 normh = 0.00;
 for(j=0;j<NUMUNITS;j++)  /* send weights to output file */
  {
  for(k=0;k<NUMUNITS;k++)
   {
   normo += wo[j][k]*wo[j][k];
   fprintf(out,"%f\n",wo[j][k]);
   }
  }
 for(j=0;j<NUMUNITS;j++)   /* send weights to output file */
  {
  for(k=0;k<NUMUNITS;k++)
   {
   normh += wh[j][k]*wh[j][k];
   fprintf(out,"%f\n",wh[j][k]);
   }
  }
 fclose(out);
 printf("\nThe norm of the output weights is");
 printf("%f",sqrt((double)normo));
 printf("\nThe norm of the hidden weights is");
 printf("%f",sqrt((double)normh));
 }
}


output(i,w,u)   /* This function computes the data transfer values for a */
          /* layer of units (0,1) */
float  i[], w[][NUMUNITS], u[];
```

```
{
int j,k;        /* counters for number of units */
double exp();   /* math function */

for(k=0;k<NUMUNITS;k++)
 {
 u[k]=0.00;

 for(j=0;j<NUMUNITS;j++)
  {
  u[k] += w[k][j] * i[j];
  }

 u[k]=1.00 / (1.00 + (float)exp(-(double)u[k])); /* threshold */
 }

}


error(o,d,energy,delta) /* This function compute first deltas and energy */
float o[], d[], *energy, delta[];

 {
 int j;
 float temp; /* scratch variable */

 *energy = 0.00;

 for(j=0;j<NUMUNITS;j++)
  {
  temp = d[j]-o[j];
  delta[j] = temp * o[j] * (1.00 - o[j]);
  *energy += temp * temp;
  }

 }


weight(delta,w,dw,i) /* This function compute new weights and deltas */
float delta[], w[][NUMUNITS], dw[][NUMUNITS], i[];

 {
 int j,k;                /* counters for units */
 float temp[NUMUNITS];         /* scratch vector for computing delta */
 float w1[NUMUNITS][NUMUNITS]; /* old values of weights */

 for(k=0;k<NUMUNITS;k++)       /* new weights */
  {
```

```
  for(j=0;j<NUMUNITS;j++)
   {
   w1[k][j] = w[k][j];
   dw[k][j] = ETA * delta[k] * i[j] + ALPHA * dw[k][j];
   w[k][j] += dw[k][j];
   }

  }

 for(k=0;k<NUMUNITS;k++)        /* new deltas */
  {
  temp[k] = 0.0;

  for(j=0;j<NUMUNITS;j++)
   {
   temp[k] += w1[k][j] * delta[j];
   }

  }

 for(k=0;k<NUMUNITS;k++)
  {
  delta[k] = i[k] * (1.0 - i[k]) * temp[k];
  }

 }
```