National Library of Canada
Acquisitions and Bibliographic Services Branch
395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada
Direction des acquisitions et des services bibliographiques
395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Canada

PERCEPTRON-BASED ALGORITHMS AND ANALYSIS

HONGFENG YIN

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

APRIL 1996

ISBN 0-612-10912-7

Abstract

Perceptron-Based Algorithms and Analysis

Hongfeng Yin, Ph.D.
Concordia University, 1996

An unsupervised Perceptron algorithm and several of its generalizations are proposed in this thesis. Under some conditions, it is proved that the unsupervised Perceptrons converge to the first principal component of the input data. Also, the convergence speed, robustness, bias and variance of a neural network learning algorithm are defined and analyzed. The learning performances of the unsupervised Perceptron algorithms, the Oja learning algorithms and the Widrow-Hoff learning algorithm are analyzed. Some simulation results and comparisons are provided. A tree classifier based on the unsupervised Perceptrons is given and applied to Chinese character recognition. In addition, an asymmetric associative memory network is proposed using the Perceptron learning algorithm. A deepening impression method is given to enhance the performance of the associative memory. Moreover, the back-propagation algorithm is improved for pruning the hidden neurons in a three layered neural network.

Acknowledgments

I am deeply indebted to my supervisor, Dr. Stan Klasa, for his capable guidance, valuable advice and critical suggestion in preparation of this thesis.

I express my appreciation to Dr. K. Khorasani for his valuable discussion.

I am grateful to Professor Ju Wei Tai and Professor Jue Wang for their encouragement and help.

I wish to thank my wife, Diyan Liu, for her patience, understanding and encouragement throughout this program.

Dedicated to my parents
Yin Zhixian and Huang Suqin

Contents

List of Figures

List of Tables

Chapter 0

Introduction

Artificial neural networks, commonly referred to as "neural networks", have been widely researched in recent years as a new information storing and processing tool [45, 58, 29, 52, 8, 88, 5]. Already, neural network models have been applied in pattern recognition, expert system, automatic control, image and signal processing and many other fileds [6, 20, 21, 27, 75, 86, 87, 92].

Research on neural networks has been motivated by the recognition that the brain computes in an entirely different way from the conventional computer. The human brain contains billions of neurons. A typical neuron has a number of ramifying branches called the dendrites; these comprise a major part of the input surface of the neuron. Axons are the transmission lines of a neuron. An axon has a smoother surface, fewer branches, and greater length. Many neurons contact thousands of other cells at sites called synapses. A typical artificial neuron or a node usually forms a weighted sum of some inputs and passes the result through a nonlinear function as shown in Fig. 1. Therefore, a neural network, which is composed of a number of connected nonlinear nodes, is a nonlinear dynamic system or a nonlinear function. The output of an artificial neuron is of the form

$$y = f(\sum_{i=0}^{N-1} w_i x_i + \theta),$$

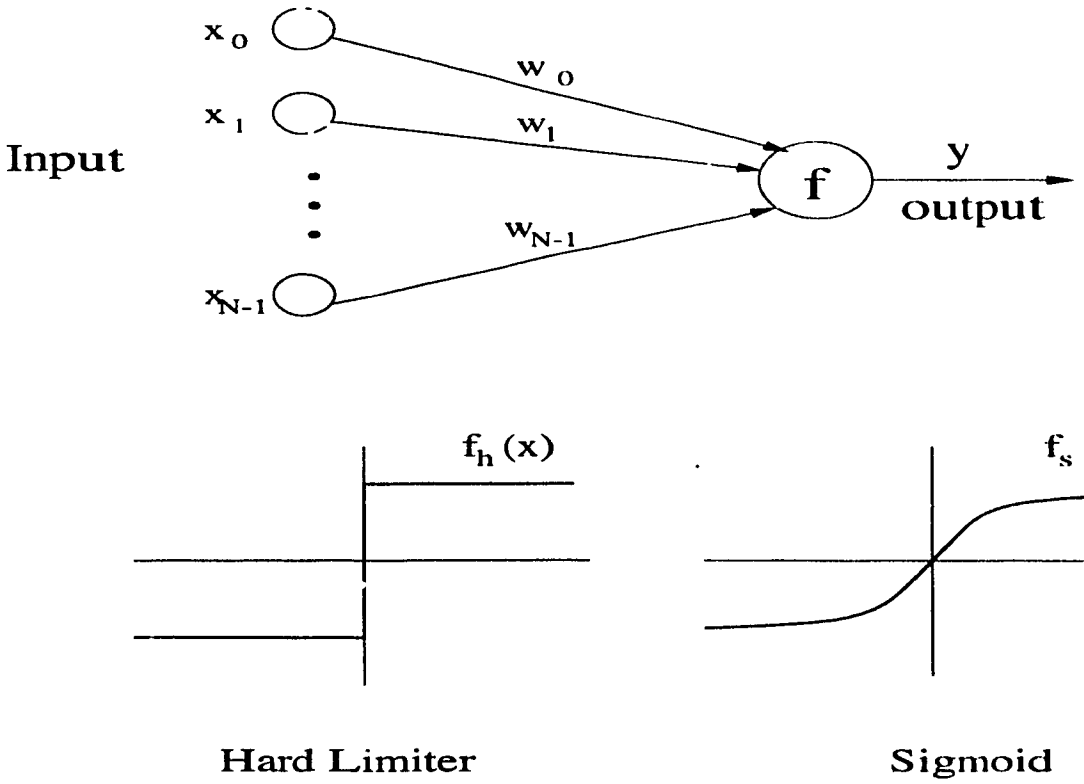

Figure 1: The structure of an artificial neuron

where w_i is a weight component, x_i is an input component and θ is a threshold. The nonlinear function f is usually a hard limiter or a sigmoid function as shown in Fig. 1. As an information processing tool, neural networks distribute information among the weighted connections and the information can be processed in a massively parallel way. Such structure provides a great degree of robustness because local damage does not impair overall performance significantly. Also, neural networks can process distorted or noise information with high performance.

From a mathematical point of view, neural networks are a new kind of nonlinear function representation method and can approximate a large class of functions. Thus, neural networks can be widely used in various fields.

In contrast to von Neumann sequential computers in which the operations and

knowledge are encoded by programs, the operations of a neural network usually follow some physical or biological rules. Neural networks can improve their own performance by self-organization, self-learning and self-adaptation.

Early research on neural networks began in the 1940s by McCulloch and Pitts [49]. They combined neurophysiology and mathematical logic using the all-or-none property of neuron firing to model the neuron as a binary discrete-time element. The McCulloch-Pitts neuron was the first model to tie the study of neural nets to the idea of computation. In the 1960s, Rosenblatt proposed a Perceptron algorithm which attracted a great number of researchers [62]. Minsky and Papert analyzed the Perceptron in detail and showed its limitations [51]. Research in neural networks came to a virtual halt in the 1970s. Recently, there has been a resurgence of interest in neural networks due to the work of Hopfield [31, 32], Rumelhart and McClelland [65], and many others [18, 39, 73]. There are several reasons for this, including the appearance of faster digital computers on which to simulate larger networks, interest in building massively parallel computers, and the discovery of powerful networks and learning algorithms.

Most neural network models have self-learning algorithms. Like the brain, a neural network can acquire knowledge through a learning process in which the weights are updated once for each sample. The neural network learning is different from the traditional statistical learning in which samples are treated as a batch.

The general neural network learning problem can be described as follows:

Let X denote a sample vector and W be the weight vector of a neural network. The ultimate purpose of a neural network learning algorithm is to optimize the expected value of a criterion $J(W, X)$. Using the hill-climbing method, the neural network

learning equation is derived as

$$W(k+1) = W(k) + \gamma_k \left. \frac{\partial J(W, X_k)}{\partial W} \right|_{W=W(k)}. \quad (1)$$

0.1 Perceptron and Linear Classifiers

The motivation of research on the Perceptron was to develop brainlike systems [62]. The Perceptron was the first neural network model which provided an automatic learning algorithm. The simplest Perceptron can be considered as a training model for a single neuron.

Assume that there are M N-dimensional sample vectors arranged as an infinite periodic sequence with period M and each sample vector is augmented into a $(N+1)$-dimensional vector with last component 1.

$$X_0, X_1, ...X_k, ...$$

Let Y_k denote the desired or the target output corresponding to X_k with binary value $+1$ or -1. Then, the Perceptron learning algorithm is summarized in the following steps:

1. Initialize the weight vector of the neuron W_0 randomly.

2. Present a new sample X_k with its desired output Y_k.

3. Modify the weight vector as:

$$W_{k+1} = \begin{cases} W_k + X_k & \text{if } W_k^T X_k \leq 0 \text{ and } Y_k \geq 0, \\ W_k - X_k & \text{if } W_k^T X_k > 0 \text{ and } Y_k < 0, \\ W_k & \text{otherwise.} \end{cases} \quad (2)$$

4. If the weight vector stays the same for M steps, the learning ends. Otherwise, go to step 2.

Rosenblatt proved that the Perceptron learning algorithm is convergent for linearly separable samples [63].

The Perceptron can be derived from the following criterion function:

$$J(W, X) = \frac{1}{2} W^T X [f_h(W^T X) - Y].$$

Also, a linear classifier can be constructed using traditional statistical methods.

Since the classification hyperplane of the Perceptron is linear in nature, the standard Perceptron algorithm cannot solve the nonlinear classification problems. However, the Perceptron remains one of the most important neural network models. These are the following main reasons:

1. The convergence is proven for linearly separable samples. This makes the analyses of the system behavior easy and explicit.

2. The Perceptron provides an important tool to explore the behavior of biological neurons.

3. The main limitation of the Perceptron is that linearly nonseparable problems cannot be directly solved. The problem can be solved by adding new nonlinear terms or new features. However, these methods are empirical and depend on the applications. The learning algorithm for growth network which can solve this problem for binary samples and for continuous samples have been proposed [85, 50, 48].

4. Baum showed that the Perceptron learning algorithm is fast for nonmalicious distributions [3]. The fast learning makes real-time and dynamic development of a system possible.

5. The samples are presented one by one and the weights are updated only according to the current sample.

6. The outputs are binary. Therefore, it is easy to represent the symbolic knowledge in the Perceptron based neural networks.

7. For binary samples, only integer operations are needed for training. As a result, all weights are integer numbers. All connection weights are initially set to zero. Only connections that are incremented during training become nonzero.

0.2 Widrow-Hoff Model

The Widrow-Hoff model is another important supervised linear neural network model which is widely used in adaptive signal processing and pattern classification. The Windrow-Hoff algorithm [79, 80], based on the least-mean-square criterion, is also called the LMS algorithm.

Fig. 2 shows the structure of a linear network. The input-output relation of the neural network is described by

$$y = \sum_{i=1}^{N} w_i x_i = W^T X,$$

where $x_N = 1$ is an augmented component for the input vector.

Let d denote the desired response or target output. The error signal is defined by

$$e = d - y.$$

As a performance measure, we introduce the mean-squared error

$$E(W) = -\frac{1}{2} E_X(e^2).$$

For the input sample X, the squared-error is

$$J(W, X) = -\frac{1}{2}(d - W^T X)^2.$$

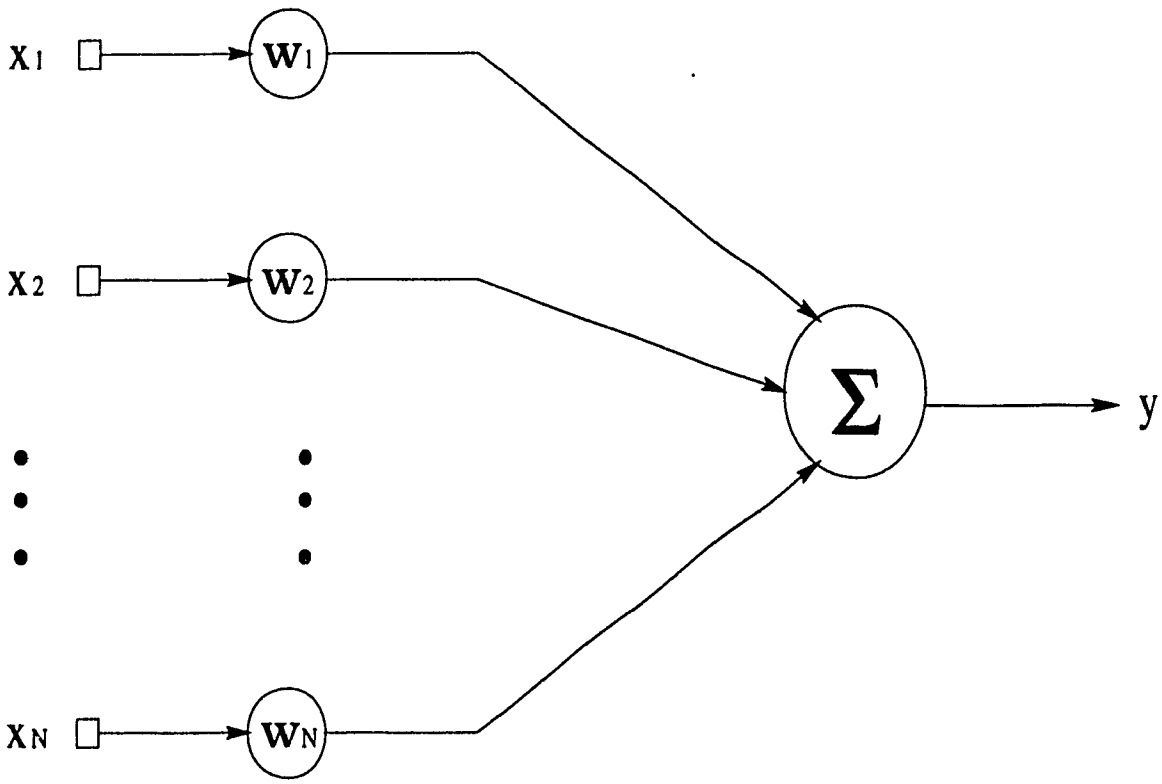

Figure 2: The structure of a linear network

Since $\frac{\partial J(W,X)}{\partial W} = (d - W^T X)X = eX$, the LMS learning equation for the constant learning rate can be derived as following based on (1):

$$W_{k+1} = W_k + \gamma(d_k - W_k^T X_k)X_k. \tag{3}$$

There are two distinct aspects of the convergence problem in the LMS algorithm:

1. The LMS algorithm is said to be convergent in the mean if the mean value of the weight vector W_k approaches the optimum solution $\overline{W}$ as k tends to infinity; that is

$$\lim_{k\to\infty} E(W_k) = \overline{W}.$$

2. The LMS algorithm is said to be convergent in the mean square if

$$\lim_{k\to\infty} E(e_k) = constant.$$

Widrow and Stearns [81] proved that the LMS algorithm is convergent in the mean if the learning-rate parameter satisfies the following condition:

$$0 < \gamma < \frac{2}{\lambda_{max}},$$

where λ_{max} is the largest eigenvalue of the autocorrelation matrix $C = E(XX^T)$. And, the LMS algorithm is convergent in the mean square if the learning-rate parameter satisfies the following condition:

$$0 < \gamma < \frac{2}{tr(C)},$$

where $tr(C)$ is the trace of C.

For the non-constant learning-rate, the convergence of the LMS algorithm will be analyzed in Chapter 3.

0.3 Hopfield Networks and Boltzmann machine

Hopfield rekindled the interest in neural nets by his extensive work on different versions of the Hopfield net [31, 32]. The Hopfield net can be used as an associative memory model. Also, the Hopfield network has been applied to solve optimization problems. The Hopfield net is a dynamic system with symmetric weights ($w_{ij} = w_{ji}$) as shown in Fig. 3. The evolution equation is of the form

$$s_i(t+1) = f(\sum_{j}^{N-1} w_{ij}s_j(t) + \theta_i), \quad 0 \le i \le N-1, \tag{4}$$

where $s_i(t)$ is the ith state of the network. f is a hard limiter or a sigmoid function.

Hopfield's contribution was to associate with such a net a measure called *energy*,

$$E = -\frac{1}{2}\sum_{ij} s_i s_j w_{ij} - \sum_i s_i\theta_i.$$

This is not the physical energy of the neural net; it is rather a mathematical quantity. If we pick a unit and the above firing rule does not change its s_i, it will not change E. However, if s_i initially equals 0 and $\sum w_{ij}s_j + \theta_i \ge 0$, then s_i goes from 0 to 1 with other s_j constant, and the energy change in E, is given by

$$\Delta E = -\sum w_{ij}s_j - \theta_i \le 0.$$

Similarly, if s_i initially equals 1, and $\sum w_{ij}s_j + \theta_i < 0$, then s_i goes from 1 to 0 with other s_j constant, and the energy gap is given by

$$\Delta E = -\sum w_{ij}s_j - \theta_i < 0.$$

In other word, on every updating we have $\Delta E \le 0$. Hence the dynamics of the net is to move to a minimum. The updating rule of (4) may not yield a passage to minimum energy for a network with asymmetric weights, but might instead yield a limit cycle, which will be discussed in Chapter 5.

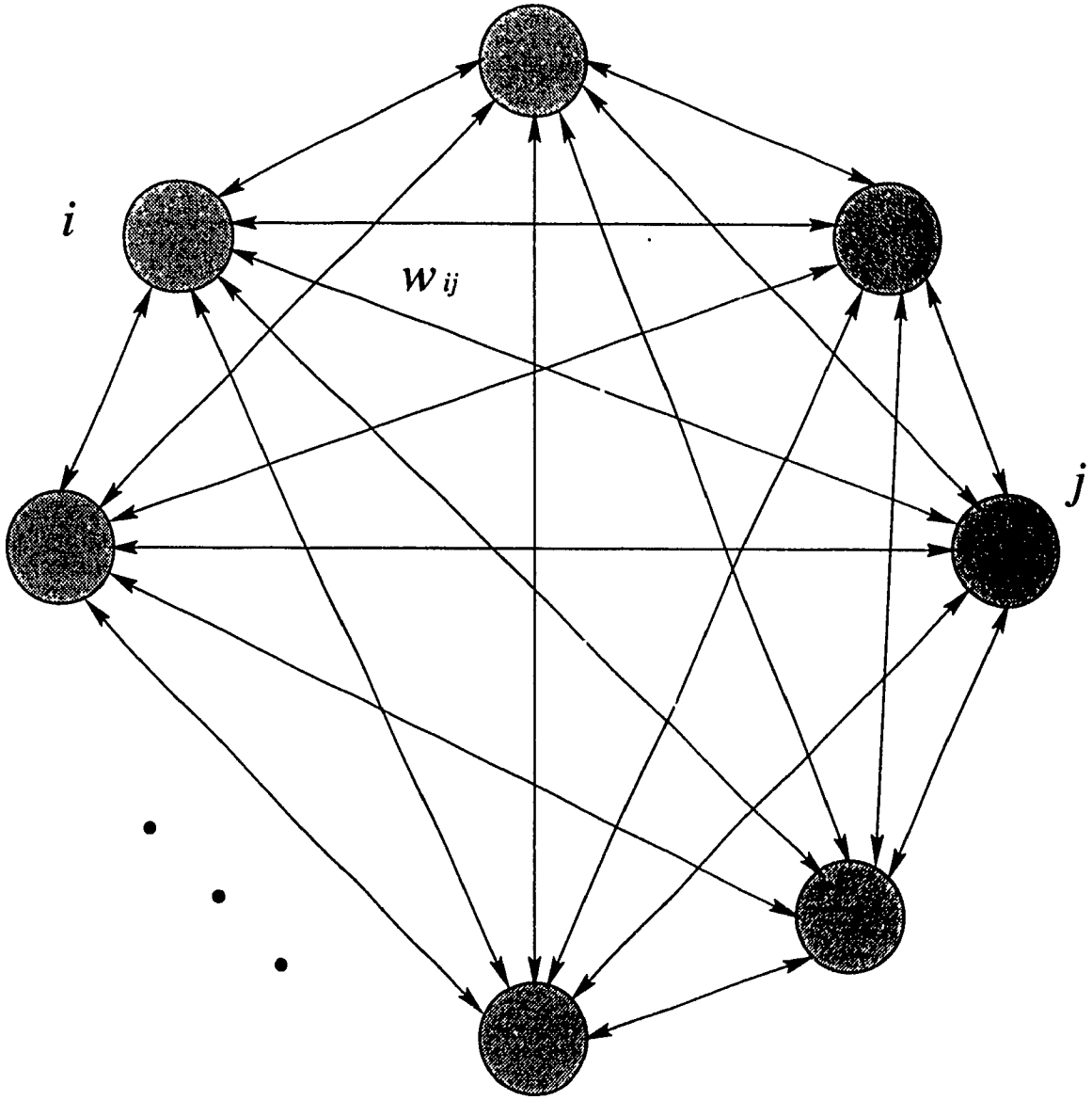

Figure 3: The structure of the Hopfield network

The Hopfield net can be used to approximately solve the famous traveling salesman problem and other optimization problems. Unfortunately, there is no guarantee to find a global minimum.

Kirkpatrick et al. [37] developed a simulated annealing method for making an optimal algorithm capable of escaping from local optima. Hinton, Sejnowski, and Ackley [30] applied simulated annealing to the Hopfield nets by modifying the rule for state transition as follows:

Compute $\Delta E = \sum s_j w_{ij} + \theta_i$.

Set s_i to 1 with probability $p_i(\Delta E) = \frac{1}{1+e^{-\Delta E/T}}$.

Annealing is a process whereby this passage to equilibrium is repeated to successively lower T. The resulting system is called a Boltzmann machine. It is proven that the system passes to its global minimum if the process is slow enough [24].

On the other hand, when a Hopfield net is used as an associative memory model, it may take the advantages of local minima. For associative memory with binary samples, the weights of the network are assigned as

$$w_{ij} = \begin{cases} \sum_{s=0}^{M-1} x_i^s x_j^s & i \neq j \\ 0 & i = j. \end{cases} \tag{5}$$

The samples may happen to be the local minima of the network with the above weights. When a corrupted input pattern is presented at iteration zero, as the net iterates, the output becomes more and more like the correct sample pattern.

The Hopfield net has two major limitations when used as a content addressable memory. First, the number of patterns that can be stored and accurately recalled is severely limited. If too many patterns are stored, the net may converge to novel spurious pattern different from all samples. Hopfield [31] showed that this occurs infrequently when samples are generated randomly and the number of samples is

less than .15 times the number of input elements or nodes in the net. The second limitation of the Hopfield net is that there is no guarantee that all samples are made to be the attractors of the network. Some improvements and applications of the Hopfield model were given in [85, 86, 72, 34].

0.4 Back-Propagation Network

We know that the main limitation of the standard Perceptron algorithm is that it can only solve linear classification problems. Also, it is well known that a three-layer Perceptron with hard limiting nonlinearity can form arbitrarily complex decision regions and a three-layer network with sigmoid nonlinearity can approximate any continuous function. The main problem in the past was that there were no effective training algorithms. This was changed by the introduction of the Back-Propagation(BP) algorithm by Rumelhart et al. The back-propagation algorithm has become the most commonly used neural network learning algorithm and is used in many applications.

The BP model is a multi-layer network with hidden layers as shown in Fig. 4. The BP training algorithm is an iterative gradient algorithm designed to minimize the mean square error between the actual output of the network and desired output. The following criterion is used:

$$E = -\sum_k (t_k - o_k)^2.$$

where k ranges over designated output units. t_k is the desired output and o_k is the actual output of node k elicited by the current input pattern.

The following sigmoid function can be used for each node:

$$f_s(x) = \frac{1}{1 + e^{-x}}.$$

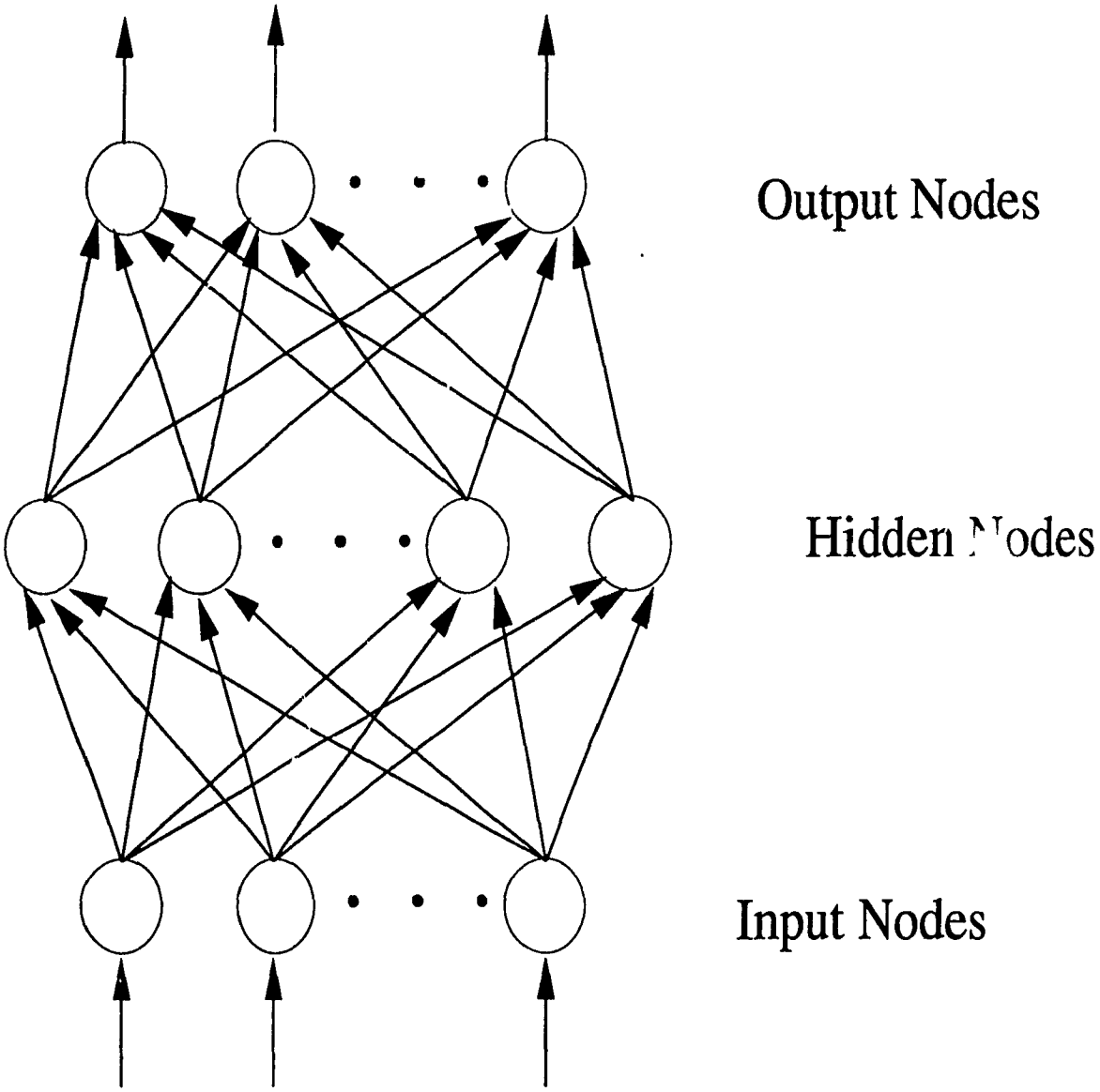

Figure 4: The neural network structure of the Back-propagation model

The back-propagation learning algorithm is summarized by the following equations. The forward propagation algorithm is defined as

$$net_{pj} = \sum_i w_{ji} o_{pi} \tag{6}$$

and

$$o_{pj} = f_s(net_{pj}). \tag{7}$$

The back-propagation algorithm is

$$\Delta_p w_{ji} = \eta \delta_{pj} o_{pi}. \tag{8}$$

The error signal is given by

$$\delta_{pj} = \begin{cases} (t_{pj} - o_{pj}) f_s'(net_{pj}) & \text{if the neuron is an output unit} \\ f_s'(net_{pj}) \sum_k \delta_{pk} w_{kj} & \text{if the neuron is not an output unit.} \end{cases} \tag{9}$$

The BP algorithm has three main limitations. Firstly, the energy surface may have many local minima, so the algorithm cannot always be guaranteed to converge to the optimal solution. The second problem is that it is difficult to analyze the behavior of hidden units in a multilayered network. It is not easy to estimate the exact number of hidden units required for a given problem before the network is trained. The third problem is that the back-propagation algorithm is often slow. Extensive surveys of the literature concerning the B-P algorithm are available in [83, 28].

0.5 Oja Models and PCA

Principal Component Analysis (abbreviated PCA) is a well-know, widely used technique, because it provides optimal linear solutions to information representation, feature extraction and data compression. The statistical PCA is perhaps the oldest and best-known technique in multivariate analysis.

There is a close correspondence between the behavior of self-organized neural networks and the statistical methods of principal component analysis. Oja first showed that a single linear neuron with a Hebbian-type adaptation rule for its synaptic weights can evolve into a filter for the first principal component of the input data [54]. Since then, various neural algorithms have been proposed to do principal component learning [55, 69, 41, 57, 36, 89, 91]. Specifically, Oja's learning equation of the stochastic gradient ascent (SGA) algorithm is of the form

$$W'_{k+1} = W_k + \gamma_k X_k X_k^T W_k, \tag{10}$$

$$W_{k+1} = W'_{k+1}/||W'_{k+1}||, \tag{11}$$

where X_k is the input vector and W_k is the weight vector. Under certain conditions, Oja proved that W_k tends almost surely to a unit eigenvector of $C = E(X_k X_k^T)$ corresponding to the largest eigenvalue.

Also, Oja and Karhunen have given the following linearized SGA algorithm to extract principal components:

$$W_{k+1} = W_k + \gamma_k[X_k X_k^T W_k - (W_k^T X_k X_k^T W_k)W_k]. \tag{12}$$

The local asymptotical properties of the ordinary differential equation corresponding to (12) were discussed in [84].

Oja's algorithms correspond to the following quadratic optimal problem: maximization of the output variance $E\{|W^T X|^2\}$ under the constraint $W^T W=1$. Since quadratic criteria weigh heavily large values that may be due to noise or outliers, the quadratic criteria are less robust than more linear criteria, such as the absolute criterion $E\{|W^T X|\}$ under the constraint that $W^T W=1$. In papers [35, 82], the robust

PCA learning algorithms were discussed. In Chapter 1, an unsupervised Perceptron algorithm is proposed and can be used for principal component analysis. Also, we will show that the unsupervised Perceptron algorithm is more robust than Oja's algorithms in the presence of outliers.

The neural network PCA methods can also be extended to extract the mth principal component. Sanger [69, 70] gave a self-organizing network consisting of a feedforward structure with a single layer of neurons, which extracts all the principal components. Another neural network model with lateral inhibitions, proposed by Kung and Diamantartas [41], can compute the $(j+1)$th principal component in an iterative manner if the first j principal components are given.

0.6 Kohonen Self-Organizing Model

Another kind of unsupervised neural network algorithms is represented by the self-organizing models based on competitive learning. The idea of competitive learning may be traced back to the works of von der Malsburg on the self-organization of orientation sensitive nerve cells in the striate cortex [77], Fukushima's cognitron [19], Grossberg on adaptive pattern classification [25], Kohonen's self-organizing feature mapping algorithm [39, 40]. For these kinds of unsupervised learning algorithms, the output neurons of the network compete among themselves to be activated or fired, with the result that only one output neuron is on at any one time. The output neurons that win the competition are called winner-take-all neurons.

The principal goal of the self-organizing feature-mapping (SOFM) algorithm is to transform a high dimensional pattern into a one- or two-dimensional discrete map as shown in Fig. 5, and to perform this transformation adaptively in a topological ordered fashion. The algorithm that forms feature maps requires a neighborhood to

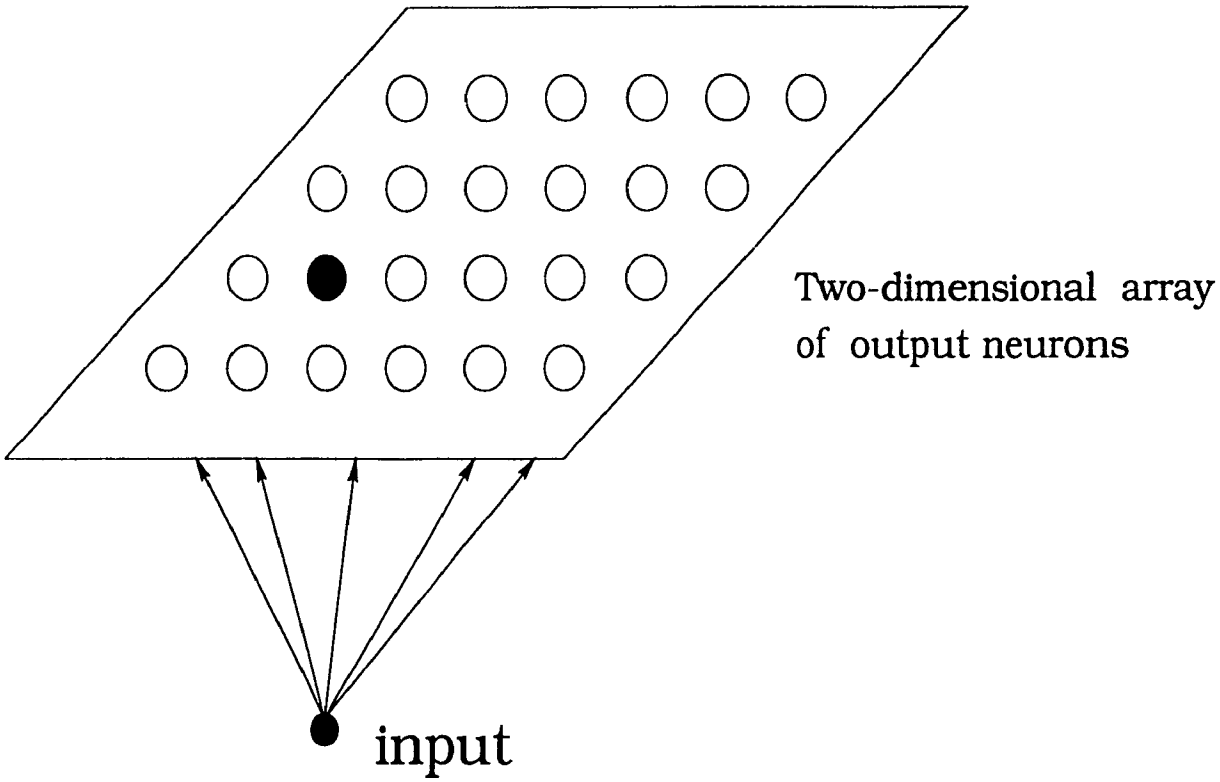

Figure 5: Kohonen's model

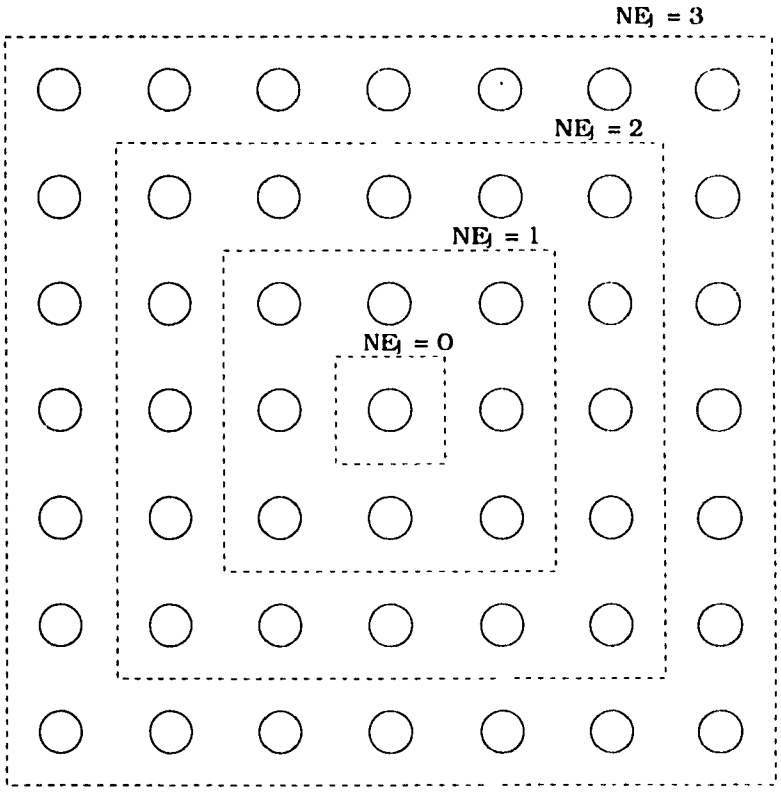

Figure 6: The neighborhood of the winning neuron j

be defined around each node as shown in Fig. 6. This neighborhood slowly decreases in size. There are three basic steps involved in the application of the algorithm after initialization, namely, sampling, similarity matching, and updating. The algorithm is given as follows:

1. **Initialization**

   For $j = 0, 1, ...M$, initialize weight vector $W_j(0)$, which is the connections from inputs to output node j. Set the initial radius of the neighborhood.

2. **Sampling**

   Draw a sample X from the input distribution with a certain probability.

3. **Similarity Matching**

   Find the best-matching (winning) neuron $i(X)$ at time n. Compute each (Euclidean) distance d_j between input and each output node j

$$d_j = ||X(n) - W_j(n)||, \qquad j = 1, 2, ..., N.$$

   Denote by $i(X)$ the node at which $d_{i(X)}$ has minimum value.

4. **Updating**

   Adjust the weight vectors of neurons, using the following formula

$$W_j(n+1) = \begin{cases} W_j(n) + \gamma_n[X(n) - W_j(n)], & j \in NE_{i(x)}(n), \\ W_j(n) & \text{otherwise.} \end{cases}$$

5. **Repeat** Go to step 2.

The performance of the SOFM algorithm is critically dependent on the selection of the main parameters, namely, the learning-rate γ_n and the radius of the neighborhood

of $NE_{l(X)}(n)$. Unfortunately, there is no theoretical basis for the selection of these parameters. They are usually determined by a process of trial and error.

The SOMF algorithm is simple to implement. The asymptotic convergence of the SOMF algorithm to a unique solution for the special case of a one-dimensional lattice was proven by Lo et al. [47] using a step by step constructive approach.

0.7 Organization of the Thesis

The purpose of this thesis is to develop some new efficient neural network models, to provide some deep theoretical analyses and to solve real application problems using these models. The Perceptron is still a very important and useful model because of its simplicity, proven convergency and extensibility. Several Perceptron based algorithms are given and analyzed in the thesis.

In Chapter 1, we present an unsupervised Perceptron algorithm and several generalizations. The convergence of the unsupervised Perceptron is proven. The idea of the unsupervised Perceptron was elicited when we developed a Chinese character recognition system using a binary tree classifier. The traditional statistical PCA method was first used to construct a hyperplane to split samples into two subsets at each node of the tree. Then, we tried to use a simple neural network algorithm to construct the hyperplane. The algorithm is the unsupervised form of the Perceptron algorithm. Therefore, it is called the unsupervised Perceptron. Next, the simulation results on 50-dimensional Chinese character samples showed that the algorithm is convergent.

Further simulations showed that the performances of the learning algorithms can be very different for different parameters, such as the learning rate, the eigenvalues

of input samples. However, there are few theoretical analyses on the learning performance in the literature. We found that the stochastic approximation theory can also be extended to analyze the learning performances. Therefore, convergence speed and robustness of learning algorithms were defined and analyzed. Also, it was demonstrated that convergence speed and robustness are highly related. The convergence speed and robustness can only measure the global performances of an algorithm. However, learning error cannot be estimated by them for a given number of learning steps. The learning error is composed of the bias and variance. The approximation of learning errors for a learning algorithm can be derived by bias and variance analysis. Therefore, an optimal learning rate can be chosen by minimizing the learning error. These analyses and formulas can provide some theoretical basis in neural network system designs and the learning parameter selections. Also, the theoretical analysis can help us understanding the deep mechanism of the neural network learning. These definitions and general analysis are described in Chapter 2.

In Chapter 3, we apply the general definitions and analyses of Chapter 2 to several single neuron based algorithms. First, convergence of the generalized and the normalized unsupervised Perceptrons is proven. The learning performance formulas for the unsupervised Perceptrons, the Oja's models and the Widrow-Hoff algorithm are explicitly derived. In the presence of outliers, both the experimental results and the theoretical analyses are presented and compared for the unsupervised Perceptrons and the Oja algorithms.

In Chapter 4, a binary tree classifier is constructed using PCA algorithms. The tree is well balanced. At each node, it is shown that the hyperplane can minimize the partitioning error. When the classifier is used for Chinese character recognition, high recognition speed and recognition rate are achieved. Also, the tree classifier is successfully applied to waveform recognition.

In Chapter 5, an asymmetric associative memory model is given using the Perceptron learning algorithm. Under some conditions, it is proven that the number of samples that can be stored in such a network is the same as the number of nodes in the network. In addition, a deepening impression method is given to enhance the performance of the associative memory.

One of the limitations of the back-propagation model is that the number of necessary hidden neurons is difficult to estimate before a network is trained. If less hidden neurons are used, the learning algorithm cannot converge or has underfitting problem. On the other hand, if more hidden neurons are given, it wastes resources and may have overfitting problem. In Chapter 6, by setting different learning rates and initial strengths of the learning weights, we obtained an improved B-P algorithm. After training, there is a big gap between the weight values of useful hidden neurons and unimportant hidden neurons. We proved that only a small error was introduced after pruning the unimportant neurons. Therefore, the algorithm can automatically find the number of useful hidden neurons. Also, the improved B-P algorithms can decrease the probability of the learning algorithms entering local minima.

Chapter 1

Unsupervised Perceptron Algorithms

1.1 Introduction

Neural network learning algorithms can be classified as being supervised or unsupervised according to the availability of an external *teacher*. The training patterns are a set of input-output examples for supervised learning algorithms. The teacher is able to provide a *desired* or *target response* for a training pattern. The *error signal* can be defined as the difference between the actual response and desired response. The learning is to adjust the weight vector to minimize the error signal in a step-by-step fashion. The criterion of the error signal usually corresponds to an absolute function or a quadratic function.

There is no external teacher in unsupervised learning algorithms. That is, the training samples do not have desired outputs. Unsupervised learning is usually based on some self-organization rules, such as the Hebbian and competitive learning rules. Like supervised learning algorithms, some unsupervised learning algorithms have also the corresponding criterion functions. Unsupervised learning algorithms are usually used to extract features and structures and to represent distributions.

For a two-layered neural network with a single output neuron, there may be four classes of learning algorithms according to the type of criterion and the availability of a teacher as shown in Table 1. They are the supervised learning algorithms with quadratic criterion (the Widrow-Hoff algorithm), the supervised learning algorithms with absolute value criterion (the Perceptron), the unsupervised learning algorithms with quadratic criterion (the Oja algorithms), and the unsupervised learning algorithms with absolute value criterion. The first three classes of learning algorithms are well known and widely used. In [89, 90, 91], we proposed and analyzed the last kind of learning algorithms: the unsupervised Perceptron algorithms. These four classes of learning algorithms are called the *single neuron based learning algorithms* in this thesis.

	Quadratic Criterion	Absolute Value Criterion
Supervised	Widrow-Hoff Algorithm	Perceptron
Unsupervised	Oja Algorithms	Unsupervised Perceptrons

Table 1: The learning algorithms for a single neuron

1.2 The Unsupervised Perceptron and Generalizations

The unsupervised Perceptron (UP) has the same structure as that of the standard Perceptron [62, 51]. But, there are no desired outputs for the input samples. The weight vector of the network can be initialized randomly. For an input sample X_k, the weight vector is modified as follows:

$$W_{k+1} = \begin{cases} W_k + X_k & \text{if } W_k^T X_k > 0, \\ W_k - X_k & \text{otherwise,} \end{cases} \tag{13}$$

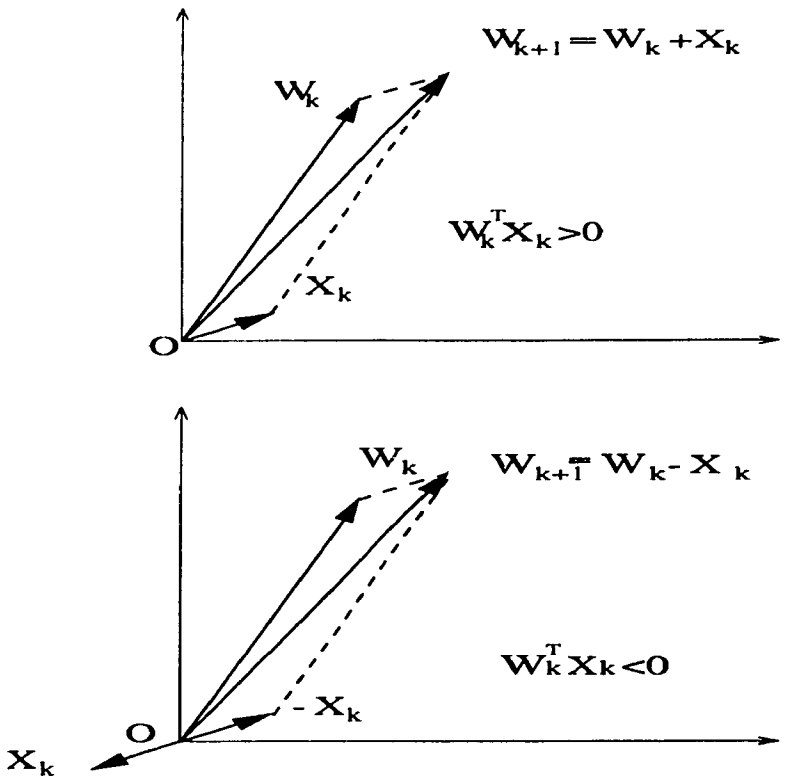

Figure 7: Geometrical interpretation of the unsupervised Perceptron algorithm

where W_k is the weight vector after kth step of learning and X_k is the kth input sample.

The geometric structure of the unsupervised Perceptron learning algorithm is shown in Figure 7.

Let

$$Y_k = \begin{cases} X_k & \text{if } W_k^T X_k > 0, \\ -X_k & \text{otherwise.} \end{cases} \tag{14}$$

Equation (13) may be written as

$$W_{k+1} = W_k + Y_k. \tag{15}$$

Let $\dot{W}_k = \frac{W_k}{k-1}$. Substituting it in (15), we can easily get the following:

$$\tilde{W}_{k+1} = \tilde{W}_k + \frac{1}{k}(Y_k - \tilde{W}_k). \tag{16}$$

Replacing $\frac{1}{k}$ by the general learning rate γ_k, we have the following generalized unsupervised Perceptron (GUP) learning algorithm:

$$W_{k+1} = W_k + \gamma_k(Y_k - W_k). \tag{17}$$

Another generalization of the UP algorithm is the following normalized unsupervised Perceptron (NUP) algorithm:

$$\tilde{W}_{k+1} = W_k + \gamma_k Y_k, \tag{18}$$

$$W_{k+1} = \frac{\tilde{W}_{k+1}}{||\tilde{W}_{k+1}||}. \tag{19}$$

The norm of the weight vector is always normalized as 1 in the NUP algorithm.

1.3 The Relationships Between Unsupervised Perceptrons and Competitive Learning Algorithms

The unsupervised Perceptron algorithms are closely related to competitive learning algorithms based on the winner-take-all rule. Competitive learning rules are widely used in unsupervised learning algorithms, such as the self-organizing feature map (SOFM) model developed by Kohonen [39], the competitive learning neural network given by Rumelhart and Zipser [67] and Grossberg's adaptive pattern classification model [25]. The winner-take-all rule permits neurons to compete for the right to respond to a given input pattern, such that only one output neuron is active at a time. Let W_j denote the weights connecting input nodes to output node j. According to the standard competitive learning rule, for an input X, the change ΔW_j applied to W_j is defined by

$$\Delta W_j = \begin{cases} \eta(X - W_j) & \text{if neuron } j \text{ wins the competition,} \\ 0 & \text{if neuron } j \text{ loses the competition.} \end{cases} \tag{20}$$

We see that the learning equation of the GUP algorithm (17) is similar to the competitive learning rule described by (20). Since the unsupervised Perceptron algorithms have only one output neuron, there is no competition for the output neuron. But, there is some competition for the direction to optimize the criterion.

We construct a competitive learning neural network with two output neurons using the following competitive rule: the neuron 1 wins if $W_1^T X > W_0^T X$; otherwise, the neuron 0 wins.

The change of the weights is defined by

$$\Delta W_j = \begin{cases} X & \text{if neuron } j \text{ wins the competition,} \\ 0 & \text{if neuron } j \text{ loses the competition.} \end{cases} \tag{21}$$

The learning equation can be obtained as

$$W_j(k+1) = \begin{cases} W_j(k) + X_k & \text{if neuron } j \text{ wins the competition,} \\ 0 & \text{if neuron } j \text{ loses the competition.} \end{cases} \tag{22}$$

Thus,

$$W_1(k+1) - W_0(k+1) = W_1(k) - W_0(k) + Y_k,$$

where

$$Y_k = \begin{cases} X_k & \text{if } (W_1(k) - W_0(k))^T X_k > 0, \\ -X_k & \text{otherwise.} \end{cases}$$

Let $W_k = W_1(k) - W_0(k)$, then

$$W_{k+1} = W_k + Y_k.$$

Therefore, this competitive learning algorithm with two output neurons is equivalent to the unsupervised Perceptron algorithm. We can consider the UP learning algorithms as special competitive learning algorithms. On the other hand, the

competitive learning algorithms can be considered as the generalized unsupervised Perceptron algorithms. Hence, the analysis of the UP algorithms can provide some basic understanding of competitive learning algorithms and give us some guidance for designing a competitive learning algorithm.

1.4 Convergence Proofs for the Unsupervised Perceptron Algorithm

In this section, we present the convergence proofs for the UP learning algorithm. The convergence analysis for the GUP and NUP learning algorithms will be given in Chapter 3.

For the unsupervised Perceptron learning algorithms, we have

Lemma 1.1 *For normally distributed samples with a covariance C and mean zero vector, assume that*

$$E(Y_k|W_k, W_{k-1}, ..., W_0) = E(Y_k|W_k),$$

where W_k is the weight vector defined by (15), or (17) or (19), and Y_k is the modification vector defined by (14). Then,

$$E(Y_k|W_k) = \rho \frac{CW_k}{\sqrt{W_k^T C W_k}}, \tag{23}$$

where $\rho = \sqrt{\frac{2}{\pi}}$.

Proof: According to (14), it is easy to verify

$$E(Y_k^T W_k|W_k) = E(X_k^T W_k|X_k^T W_k > 0).$$

Since X_k is a random vector with multivariate normal distribution, $X_k^T W_k$ is a normally distributed random variable with variance $W_k^T C W_k$. So,

$$E(X_k^T W_k | X_k^T W_k > 0) = \frac{1}{\sqrt{2\pi W_k^T C W_k}} \int_0^\infty exp(-\frac{x^2}{2W_k^T C W_k}) dx = \rho\sqrt{W_k^T C W_k}. \tag{24}$$

Then, it follows that

$$E(Y_k|W_k) = \frac{\partial E(Y_k^T W_k|W_k)}{\partial W_k} = \frac{\partial E(X_k^T W_k|X_k^T W_k > 0)}{\partial W_k} = \rho \frac{CW_k}{\sqrt{W_k^T C W_k}}.$$

Q. E. D.

Since C is positive definite, there exists an orthogonal transform matrix B, such that

$$B^T C B = D, \tag{25}$$

and

$$D = \begin{pmatrix} \lambda_1 & 0 & ... & 0 \\ 0 & \lambda_2 & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & \lambda_n \end{pmatrix},$$

where $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n \geq 0$ are the eigenvalues of C. Let $B = (B_1, B_2, ..., B_n)$. B_i is the ith unit eigenvector of C. B_1 is the unit eigenvector of C corresponding to the largest eigenvalue. It is easy to verify that vector $E(Y_k|W_k)$ and $E(W_k + Y_k|W_k)$ are closer to B_1 than W_k. Therefore, it is expected that the weight vector converges to the first eigenvector using the unsupervised Perceptron learning algorithm. Actually, we have the following theorem.

Theorem 1.1 *In algorithm (13), assume that the samples are normally distributed random vectors with a covariance matrix C and mean zero vector, and $\lambda_1 > \lambda_2$. Then, for any initial weight vector W_0, $\frac{W_k}{k}$ converges almost surely to $\rho\sqrt{\lambda_1}B_1$ or $-\rho\sqrt{\lambda_1}B_1$.*

To prove Theorem 1.1, we use the stochastic approximation theory given by Kushner and Clark that we put in Appendix. Analogous to Oja's proofs of SGA [56], first, three lemmas are proved.

Lemma 1.2 *Let the assumptions in Theorem 1.1 hold and $\overline{Z}$ be a locally asymptotically stable (in the sense of Liapunov) solution to the differential equation*

$$\frac{dZ}{dt} = \rho \frac{CZ}{\sqrt{Z^T AZ}} - Z, \tag{26}$$

with domain of attraction $D(\overline{Z})$. If there is a compact set $A \subset D(\overline{Z})$ such that the solution of (13) satisfies $P(\frac{W_k}{k} \in A$ infinitely often$)=1$, then $\frac{W_k}{k}$ tends to $\overline{Z}$ almost surely.

Proof: Let $\overline{W}_k = \frac{W_k}{k} (k = 1, 2, ...)$, the equation (13) can be written as:

$$\overline{W}_{k+1} = \overline{W}_k + \frac{1}{k+1}(Y_k - \overline{W}_k). \tag{27}$$

Since $E(Y_k|\overline{W}_k) = \rho \frac{C\overline{W}_k}{\sqrt{\overline{W}_k^T C \overline{W}_k}}$, (27) can be rewritten as:

$$\overline{W}_{k+1} = \overline{W}_k + \frac{1}{k+1}[\rho \frac{C\overline{W}_k}{\sqrt{\overline{W}_k^T C \overline{W}_k}} - \overline{W}_k] + \frac{1}{k+1}[Y_k - \rho \frac{C\overline{W}_k}{\sqrt{\overline{W}_k^T C \overline{W}_k}}]. \tag{28}$$

Consider the assumptions in Appendix. The condition i) follows with

$$h(Z) = \rho \frac{CZ}{\sqrt{Z^T CZ}} - Z.$$

Since $A_k = 0$, the condition ii) is satisfied. $\gamma_k = \frac{1}{k+1}$ implies condition iii).

Condition iv) in Appendix is verified as follows:

Let $D_k = Y_k - \rho \frac{C\overline{W}_k}{\sqrt{\overline{W}_k^T C \overline{W}_k}}$. Y_k is a function of $\overline{W}_k, X_k$. We have

$$E(D_k|D_{k-1}, D_{k-2}, ...) = 0.$$

Also,

$$E(||D_k||^2) = E(D_k^T D_k) \leq 2[E(Y_k^T Y_k) + \rho^2 E(\frac{\overline{W}_k^T C^T C \overline{W}_k}{\overline{W}_k^T C \overline{W}_k})] \leq 2(\lambda_1 + \lambda_2 + ... + \lambda_N + \rho^2 \lambda_1).$$

Thus, $\sum_{i=k}^{m} \frac{1}{i+1} D_i$ is a martingale sequence. If $\{G_m\}$ is a martingale sequence, then there is an important martingale inequality of Doob [16] which states that

$$P(\sup_{m \geq 0} |G_m| \geq \epsilon) \leq \lim_m E|G_m|^2/\epsilon^2.$$

Applying this to our problem yields

$$P(\sup_{m \geq n} |\sum_{i=k}^{m} \frac{1}{i+1} D_i| \geq \epsilon) \leq constant \sum_{i=k}^{\infty} \frac{1}{(i+1)^2}/\epsilon^2,$$

whose right side goes to zero as $k \rightarrow \infty$. Therefore, condition iv) in Appendix holds.

Lemma 1.2 follows directly from Theorem .1 of Appendix.

Then, we show that $\rho\sqrt{\lambda_1} B_1$ and $-\rho\sqrt{\lambda_1} B_1$ are asymptotically stable points of (26).

Lemma 1.3 *Let B_1 be one of the two unit eigenvectors corresponding to the largest eigenvector of matrix C. Then, the points $\rho\sqrt{\lambda_1} B_1$ and $-\rho\sqrt{\lambda_1} B_1$ are locally asymptotically stable to (25). The domain of attraction of the two points are respectively $D(B_1) = \{X \in \mathbb{R}^N | X^T B_1 > 0\}$ and $D(-B_1) = \{X \in \mathbb{R}^N | X^T B_1 < 0\}$.*

Proof: According to (25), $B^T C B = D$. Let $Z = BY$, the equation (26) can be written as

$$\frac{dY}{dt} = \rho \frac{DY}{\sqrt{Y^T DY}} - Y.$$

That is, for $i = 1, ..., n$

$$\frac{dy_i}{dt} = \rho \frac{\lambda_i y_i}{\sqrt{Y^T DY}} - y_i.$$

It follows that

$$\frac{\frac{dy_i}{dt} + y_i}{\frac{dy_1}{dt} + y_1} = \frac{\lambda_i y_i}{\lambda_1 y_1},$$

which can be rewritten as

$$\lambda_1 \frac{dy_i}{y_i} - \lambda_i \frac{dy_1}{y_1} = (\lambda_i - \lambda_1)dt,$$

whose solution under $y_1(0) \neq 0$ is

$$\frac{y_i^{\lambda_1}(t)}{y_1^{\lambda_i}(t)} = e^{(\lambda_i - \lambda_1)t} \frac{y_i^{\lambda_1}(0)}{y_1^{\lambda_i}(0)}. \tag{29}$$

For $i \neq 1$, it is easy to verify

$$\lim_{t \to \infty} \frac{y_i(t)}{y_1(t)} = 0. \tag{30}$$

Consider the equation

$$\frac{dy_1}{dt} = \rho \frac{\lambda_1 y_1}{\sqrt{Y^T DY}} - y_1.$$

For $y_1(0) > 0$, it follows that

$$\frac{dy_1}{dt} = \rho\sqrt{\lambda_1} - y_1 - m(t), \tag{31}$$

where $m(t) = \rho \frac{\lambda_1 y_1(t)}{\sqrt{Y^T DY}} - \rho\sqrt{\lambda_1}$. According to (30), it is not difficult to verify that $\lim_{t \to \infty} m(t) = 0$. The solution to (31) is

$$y_1(t) = \rho\sqrt{\lambda_1} + e^{-t}(y_1(0) - \int_0^t e^x m(x)dx).$$

Since

$$\lim_{t \to \infty} e^{-t} \int_0^t e^x m(x)dx = 0,$$

it implies that

$$\lim_{t \to \infty} y_1(t) = \rho\sqrt{\lambda_1}. \tag{32}$$

Similarly, for $y_1(0) < 0$, we have

$$\lim_{t \to \infty} y_1(t) = -\rho\sqrt{\lambda_1}. \tag{33}$$

For $i \neq 1$, according to (30)

$$\lim_{t\to\infty} y_i(t) = 0.$$

Therefore, the proof is completed.

Remark: according to (29), we have

$$y_i(t) = [y_1^{\lambda_i}(t)e^{(\lambda_i-\lambda_1)t}\frac{y_i^{\lambda_1}(0)}{y_1^{\lambda_i}(0)}]^{\frac{1}{\lambda_1}}. \tag{34}$$

Considering (32) and (34), we obtain

$$y_i(t) = O(e^{\frac{\lambda_i-\lambda_1}{\lambda_1}t}) = O(e^{(\eta_i-1)t}). \tag{35}$$

where $\eta_i = \frac{\lambda_i}{\lambda_1}$. Therefore, the convergence speed of $y_i(t)$ is $O(e^{(\eta_i-1)t})$ which mainly depends on η_i. According to (32) and (34), we also have

$$y_i(t) \approx y_i(0)(\frac{\rho\sqrt{\lambda_1}}{y_1(0)})^{\frac{\lambda_i}{\lambda_1}}e^{(\eta_i-1)t} = B_i^T Z(0)(\frac{\rho\sqrt{\lambda_1}}{B_1^T Z(0)})^{\frac{\lambda_i}{\lambda_1}}e^{(\eta_i-1)t}, \tag{36}$$

where $Z(0)$ is the initial vector of $Z(t)$.

Then, we show that $\overline{W}_k$ visits infinitely often a compact subset of the domain of attraction of one of the asymptotically stable solutions to (26).

Lemma 1.4 *Let the assumptions in Theorem 1.1 hold. Then, there exists a, such that in the stochastic process defined by (27), the event* $||\overline{W}_k|| \leq a$ *occurs infinitely often almost surely.*

Proof: For an $a > 0$, the set $A = \{X : ||X|| \leq a\}$ is a compact subset of $D(S)$. It is easy to show that $\lim_{k\to\infty} E(||\overline{W}_k||)$ is bounded by a number b. Let a be sufficiently large so that $a > b$. Then, $\{\overline{W}_k\}$ enters A infinitely often. This result can be

verified as following: if it is not true, $\{\overline{W}_k\}$ enters A a finite number of times. Then, $\lim_{k\to\infty} E(||\overline{W}_k||) > a$, which is a contradiction.

Therefore, the region A is reached by the process $\overline{W}_k$ infinitely often with probability one.

Consequently, Theorem 1.1 is proved directly according to the above three lemmas.

Proof of Theorem 1.1: According to Lemma 1.3, $\rho\sqrt{\lambda_1}B_1$ and $-\rho\sqrt{\lambda_1}B_1$ are locally asymptotically stable solutions to (26). According to Lemma 1.4, $\overline{W}_k$ visits almost surely a compact subset of the domain of attraction of point $\rho\sqrt{\lambda_1}B_1$ or $-\rho\sqrt{\lambda_1}B_1$. Then, according to Lemma 1.2, $\overline{W}_k$ converges almost surely to either $\rho\sqrt{\lambda_1}B_1$ or $-\rho\sqrt{\lambda_1}B_1$.

Theorem 1.1 shows that the trained weight vector converges to the first principal component for normally distributed samples. According to the proof of the above theorem, the convergence speed of the learning algorithm mainly depends on $\eta = \frac{\lambda_2}{\lambda_1}$. In Chapter 3, the performance of UP algorithm will be further analyzed.

According to Theorem 1.1, the largest eigenvalue can also be learned using (13). Let $\tilde{B}_1$ and $\tilde{\lambda}_1$ be the estimations of B_1 and λ_1, respectively. We have

$$\frac{W_k}{k} = \rho\sqrt{\tilde{\lambda}_1}\tilde{B}_1$$

Since $\tilde{B}_1$ is a unit vector, $\tilde{B}_1 = W_k/||W_k||$. Therefore,

$$\tilde{\lambda}_1 = \frac{||W_k||^2}{(k\rho)^2}.$$

1.5 Learning Difference Equation Analysis

According to (28), the UP algorithm corresponds to the following difference equation:

$$V_{k+1} = V_k + \frac{1}{k+1}[\rho\frac{CV_k}{\sqrt{V_k^T CV_k}} - V_k]. \tag{37}$$

Let $\gamma_k = \frac{1}{k+1}$ and $U_k = BV_k$. Then, equation (37) can be rewritten as

$$U_{k+1} = U_k + \gamma_k(\rho\frac{DU_k}{\sqrt{U_k^T DU_k}} - U_k), \tag{38}$$

where $D = diag(\lambda_1, \lambda_2, ..., \lambda_n)$. $U_i = (u_i^1, u_i^2, ..., u_i^N)$ is a N-dimensional vector. Equation (38) can be written as

$$u_{k+1}^i = (1 - \gamma_k + \gamma_k\rho\frac{\lambda_i}{\sqrt{U_k^T DU_k}})u_k^i \quad (i = 1, 2, ...N). \tag{39}$$

Using (39) recursively, we have

$$u_{k+1}^i = u_0^i \prod_{m=1}^{k}(1 - \gamma_m + \gamma_m\rho\frac{\lambda_i}{\sqrt{U_m^T DU_m}}). \tag{40}$$

Applying the logarithm operator to both sides of (40), we obtain

$$\ln u_{k+1}^i = \sum_{m=1}^{k}\ln(1 - \gamma_m + \gamma_m\rho\frac{\lambda_i}{\sqrt{U_m^T DU_m}}) + \ln u_0^i. \tag{41}$$

For $i > 1$, let $S_{k+1} = \ln u_{k+1}^i$, $a_m = \ln(1 - \gamma_m + \gamma_m\rho\frac{\lambda_i}{\sqrt{U_m^T DU_m}})$ and $a_0 = \ln u_0^i$. We rewrite (41) as

$$S_{k+1} = \sum_{m=0}^{k} a_m.$$

Since $U_m \to \rho\sqrt{\lambda_1}B_1$ or $U_m \to -\rho\sqrt{\lambda_1}B_1$, it is obvious that $U_m^T DU_m \to (\rho\lambda_1)^2$. Let us consider the limit

$$\lim_{m\to\infty}\frac{a_m}{\gamma_m} = \lim_{m\to\infty}\frac{\ln(1 - \gamma_m + \gamma_m\rho\frac{\lambda_i}{\sqrt{U_m^T DU_m}})}{\gamma_m} = \lim_{m\to\infty}\frac{-\gamma_m + \gamma_m\rho\frac{\lambda_i}{\sqrt{U_m^T DU_m}}}{\gamma_m}$$

$$= \lim_{m\to\infty}(-1 + \rho\frac{\lambda_i}{\sqrt{U_m^T DU_m}}) = -1 + \frac{\lambda_i}{\lambda_1} = -1 + \eta_i,$$

where $\eta_i = \frac{\lambda_i}{\lambda_1} < 1$. Since $\sum_{n=1}^{\infty} \gamma_n = \infty$, it is not difficult to prove

$$\lim_{n\to\infty} \frac{S_n}{\sum_{m=1}^{n} \gamma_m} = \lim_{m\to\infty} \frac{a_m}{\gamma_m} = -1 + \eta_i.$$

Since $\gamma_k = \frac{1}{k+1}$, it is known that

$$\lim_{n\to\infty} \frac{\ln n}{\sum_{m=1}^{n} \gamma_m} = 1.$$

So, we have

$$\lim_{n\to\infty} \frac{\ln u_{n+1}^i}{\ln n} = \lim_{n\to\infty} \frac{S_n}{\ln n} = \lim_{n\to\infty} \frac{S_n}{\sum_{m=1}^{n} \gamma_m} = -1 + \eta_i. \tag{42}$$

Equation (42) means that the convergence speed of $\{U_k\}$ at ith eigenvector direction is

$$u_n^i = O(n^{\eta_i - 1}) \qquad (i = 2, 3, \ldots N). \tag{43}$$

Comparing (35) with (43), we see that the convergence speed of $y_i(t)$ and the convergence speed of u_n^i are not the same, but are highly related. The more detailed analysis about their relationship will be provided in the next chapter.

1.6 Summary

In this chapter, an unsupervised Perceptron learning algorithm (UP) and several generalized UP algorithms have been proposed. For the normally distributed input patterns, we proved that the UP algorithm converges to the first principal component. Also, we analyzed the corresponding ordinary differential equation and difference equation of the UP algorithm and show that the convergence speeds of the solutions to these equations depend only on the eigenvalues of the input patterns. More definitions and detailed analysis will be presented in the next two chapters.

Chapter 2

Convergence and Learning Performance Analysis of Neural Network Algorithms

2.1 Introduction

The asymptotic behavior analysis for neural network based learning algorithms is very important and usually quite difficult to undertake, due to the fact that these algorithms are not only stochastic but also nonlinear. The stochastic approximation method, originally proposed by Robbins and Monro [61], provides a general tool for analyzing a great variety of neural network learning algorithms. By associating an ordinary differential equation to a stochastic approximation algorithm, Ljung [46] and Kushner and Clark [42] extended the results for Robbins-Monro-like methods considerably. It has been used to prove the convergency of several unsupervised learning algorithms for principal component analysis [53, 69]. In Chapter 1, we analyzed the convergence of the UP algorithm using stochastic approximation theory. However, most of these analyses focus on the convergency of the algorithms. Little attention is paid to the other important features of the learning performance, such as learning

speed, robustness, learning error, etc.

In this chapter, first, we analyzed the convergency of single neuron based learning algorithms. Then, the convergence speed, robustness, bias and variance of stochastic approximation algorithms are defined and analyzed. The formulae of these performances are derived for single neuron based learning algorithms. The analysis provides a theoretical basis for selecting the learning parameters.

2.2 Convergence Analysis of Neural Network Learning Algorithms

In the Appendix, Theorem .1 (Theorem 2.3.1 in [42]) is presented. It provides a powerful and convenient tool for analyzing the convergence of stochastic learning algorithms. Neural network learning algorithms can be considered as a special type of stochastic approximation algorithms. Based on Theorem .1, we can analyze the asymptotic behaviors of neural network learning algorithms.

Most of neural network learning algorithms can be derived from a criterion $J(W, X)$, where W is the weight vector and X is the input vector. If we take the partial derivative of $J(W, X)$ with respect to W, we obtain

$$\tilde{h}(W_k, X_k) = \left.\frac{\partial J(W, X_k)}{\partial W}\right|_{W=W_k}. \tag{44}$$

$h()$ is a vector function. Using the hill-climbing method, the neural network learning equation is derived as

$$W(k+1) = W(k) + \gamma_k \tilde{h}(W_k, X_k). \tag{45}$$

For neural network learning algorithms, since the modification of weight vector usually only depends on the current weight vector, we can assume that

$$E_X[\tilde{h}(W_k, X_k)|W_k] = E_X[\tilde{h}(W_k, X_k)|W_0, ...W_k; X_0, ..., X_{k-1}].$$

Also, we assume that $\{X_k\}$ is stationary.

Let $h(W_k) = E_X[\tilde{h}(W_k, X_k)|W_k]$. Equation (45) can also be written in the form

$$W_{k+1} = W_k + \gamma_k(h(W_k) + D_k), \tag{46}$$

where $D_k = \tilde{h}(W_k, X_k) - h(W_k)$. Without ambiguity, we may simply use $E[\tilde{h}(W_k, X_k)]$ to represent $E_X[\tilde{h}(W_k, X_k)|W_k]$.

The equations (45) and (46) are called the learning difference equation (LDE) of learning algorithms. The corresponding ordinary differential equation (ODE) of the learning algorithm is

$$\frac{dZ}{dt} = h(Z). \tag{47}$$

The convergence of W_k is described in Theorem .1.

For a criterion based algorithm, let $V(Z) = E_X(J(Z, X))$, thus

$$\frac{d}{dt}V(Z) = E_X(\frac{\partial J(Z, X)}{\partial Z})\frac{dZ}{dt} = h(Z)^T h(Z) = ||h(Z)||^2, \tag{48}$$

which means $\frac{d}{dt}V(Z)$ is a positive semidefinite function. $V(Z)$ is a Liapunov function of (47). Thus, the stationary points of (47), together with the point (∞), form an invariant set S with a global domain of attraction $D(s) = \mathbb{R}^N$.

For $a > 0$, the set $A = \{X : ||X|| \leq a\}$ is a compact subset of $D(S)$. Assume that $E(||W_k||)$ is bounded by b. Let a be sufficiently large so that $a > b$. Then, $\{W_k\}$ enters A infinitely often. This result can be verified as following: if it is not true, $\{W_k\}$ enters A a finite number of times. Then, $\lim_{k-\infty} E(||W_k||) > a$. There is a contradiction.

Also, W_k cannot tend to infinity if $E(||W_k||)$ is bounded.

Therefore, similar to Corollary 2 in [16], we have the following corollary to the convergence Theorem .1 for criterion based neural network algorithms.

Corollary 2.1 *Suppose that the attraction basin* $D(S)=\mathbb{R}^N$ *and that* $h(Z)$ *and* $V(Z)$ *have isolated stationary points. Assume that* $E(\|W_k\|)$ *is bounded. Then,* W_k *tends to a local maximum of* $V(Z)$ *almost surely.*

For single neuron based learning algorithms, the global behaviors are similar to the local behaviors in the neighborhood of stable states. The local properties of (47) can be described by a linear system $\frac{dX}{dt} = HX$, where $H = \frac{\partial h(Z)}{\partial Z}\Big|_{Z=\overline{Z}}$. If H is negative definite, the convergence speed of the solution to the linear system $\frac{dX}{dt} = HX$ is determined by its largest eigenvalue λ_h, such as $O(e^{\lambda_h t})$. For linear systems, the local convergence speed is also the global convergence speed. Usually, the ordinary differential equation (ODE) of a single neuron based learning algorithm is a kind of nearly-linear equation.

2.3 Definitions of Learning Speed and Robustness

Equation (46) also corresponds to the following difference equation:

$$U_{k+1} = U_k + \gamma_k h(U_k), \tag{49}$$

which is called the difference equation (DE) of a learning algorithm. In the following analysis, assume that $\overline{Z} \in S$ is the limit of W_k. By viewing the DE as the LDE with $D_k = 0$, we also have $U_k \to \overline{Z}$ as $k \to \infty$. That is, $(W_k - U_k) \to 0$ as $k \to \infty$ almost surely. The solutions of LDE (46) and DE (49) are highly related. The total variance introduced in learning is

$$V = \sum_{k=1}^{\infty} \gamma_k^2 E(||D_k||^2).$$

If the modified conditions iii) and iv) of Theorem .1 hold, V is bounded. For a small V, the solutions of the two equations are very close. The solutions to LDE and DE are the same if $V = 0$. Therefore, the *convergence speed* of a stochastic learning algorithm is defined as the convergence speed of the solution of its DE to an asymptotically stable state, and the ***robustness*** is defined as

$$R = \frac{||\overline{Z}||^2}{V} = \frac{||\overline{Z}||^2}{\sum_{k=1}^{\infty} \gamma_k^2 E(||D_k||^2)},$$

where $\overline{Z}$ is a locally asymptotically stable state to the ODE.

If the modified conditions iii) and iv) of Theorem .1 hold, $R > 0$.

If $||Z(t) - \overline{Z}|| = O(f(t))$, $O(f(t))$ is defined as the convergence speed for the ODE (47).

In Chapter 1, we derived the convergence speed of the solution to the DE of the UP algorithm. We see that the convergence speed of the solution to the ODE is much faster than the convergence speed of the solution to the DE. Besides, the learning rate is one of the important parameters which determine the learning speed. However, the ODE does not include this parameter. It seems that the convergence speed for the ODE is not equivalent to the convergence speed for the DE.

By further analyzing and comparing the solutions to the DE and ODE, we found that the convergence speeds of $U_k \to \overline{Z}$ and $Z(t) \to \overline{Z}$ are also highly related.

Let us assume that $\{\gamma_k\}$ satisfies (153). For a one dimensional ODE and DE, we obtain

Theorem 2.1 *Suppose that the assumptions of Theorem .1 hold. Assume that* $||h(Z(t))||$ *is a monotonically decreasing function, and* $||\frac{\partial h(Z)}{\partial Z}||$ *is bounded.* $Z(t)$ *and* U_n *are the*

solutions to the ODE (47) and the DE (49), respectively. We have, $||Z(t) - \bar{Z}|| = O(f(t))$ *is equivalent to* $||U_n - \bar{Z}|| = O(f(t_n))$, *where* $t_{n+1} = \sum_{i=1}^{n} \gamma_i$.

Proof: For a one dimensional function, using the truncated Taylor's series expansion, we have

$$Z(t_{k+1}) = Z(t_k) + (t_{k+1} - t_k) \left.\frac{dZ(t)}{dt}\right|_{t=t_k} + \frac{(t_{k+1} - t_k)^2}{2} \left.\frac{d^2 Z}{dt^2}\right|_{t=t_k+s_k}, \tag{50}$$

where $0 \leq s_k \leq \gamma_k$. Note that $\gamma_k = t_{k+1} - t_k$ and apply the ODE to (50), we obtain

$$Z(t_{k+1}) = Z(t_k) + \gamma_k h(Z(t_k)) + \frac{\gamma_k^2}{2} h'(t_k + s_k). \tag{51}$$

Let $Z_k = Z(t_k)$. Equation (51) can be rewritten as the following difference equation:

$$Z_{k+1} = Z_k + \gamma h(Z_k) + \frac{\gamma_k^2}{2} h'(t_k + s_k). \tag{52}$$

Note that $h'(t) = \frac{dh(Z)}{dZ}\frac{dZ}{dt} = \frac{dh(Z)}{dZ} h(t)$. Let $h(t_k + s_k) = G_k h(t_k) = G_k h(Z_k)$, and $M_k = \left.\frac{dh(Z)}{dZ}\right|_{Z=Z_k}$. Equation (52) is rewritten as

$$Z_{k+1} = Z_k + (\gamma_k + \frac{\gamma_k^2}{2} G_k M_k) h(Z_k). \tag{53}$$

If $||Z(t) - \bar{Z}|| = O(f(t))$, the convergence speed of Z_n is $O(f(t_n))$. Comparing (53) with the DE (49), we have that the convergence speed of U_n is $O(f(t_n - p_n))$, where $p_n = \sum_{k=1}^{n} \frac{\gamma_k^2}{2} G_k M_k$. Since G_k and M_k are bounded, then p_n is also bounded. Therefore, the convergence speed of U_n is also $O(f(t_n))$.

Similarly, it can be verified that if the convergence speed of U_n is $O(f(t_n))$, then $||Z(t) - \bar{Z}|| = O(f(t))$.

Without special claim, all the learning rates in this thesis are assumed with the form $\gamma_k = \frac{c}{k+K}$, where c is a constant and K is a large constant.

Usually, $O(f(t))$ is an exponential function with the form $f(t) = e^{-ct}$. Then, $t_n = \sum_{k=1}^{n} \gamma_k = c \ln n + O(1)$. The convergence speed of t_n is

$$O(f(t_n)) = O(f(c \ln n)) = O(e^{-c\beta \ln n}) = O(n^{-c\beta}).$$

Generally, the conclusions of Theorem 2.1 also hold for the learning algorithms in N-dimensional space. The conditions and the proofs are remain to be explored.

2.4 Definitions of Bias and Variance

The convergence speed and robustness analysis can show the global performances of a learning algorithm. Also, we need to estimate the learning error for a given number of learning steps, so that the learning error can be minimized by choosing an optimal learning parameter.

In [23], the definitions of bias and variance were given and discussed based on criteria and approximating functions of learning algorithms. Here, we give some definitions and analyses based on the stochastic approximation learning equation and the learning weight vectors.

First, the *learning error* of an algorithm is defined as the mean square error between the weight vector W_k and its limit $\overline{W}$, as shown by

$$E_X(||W_k - \overline{W}||^2 \mid X_0, X_1, ..., X_{k-1}).$$

Without ambiguity, the learning error is simply written as $E(||W_k - \overline{W}||^2)$. We have

$$E(||W_k - \overline{W}||^2) = E(||W_k - EW_k + EW_k - \overline{W}||^2)$$

$$= E(||W_k - EW_k||^2) + 2E[(W_k - EW_k)^T(EW_k - \overline{W})] + E(||EW_k - \overline{W}||^2). \quad (54)$$

Since

$$E[(W_k - EW_k)^T(EW_k - \overline{W})] = [E(W_k - EW_k)^T] \cdot [(EW_k - \overline{W})] = 0$$

and

$$E(||EW_k - \overline{W}||^2) = ||EW_k - \overline{W}||^2.$$

Eq. (54) is written as

$$E(||W_k - \overline{W}||^2) = E(||W_k - EW_k||^2) + ||EW_k - \overline{W}||^2. \tag{55}$$

Therefore, the learning error of an algorithm are composed of two parts. The mean square error between the learning weight vector and its expectation, the first term of the right hand side of (55), is defined as the *variance* of the learning algorithm at the kth step. That is,

$$Var_k = E(||W_k - EW_k||^2). \tag{56}$$

The difference between the expectation of the weight vector at the kth step and the limit of the weight vector is defined as the *bias* of the learning algorithm at the kth step. Specifically,

$$Bias_k = ||EW_k - \overline{W}||. \tag{57}$$

The diagram interpretation of the bias and variance is shown in Fig. 8.

It is difficult to derive an exact expression for the variance. For the learning rate $\gamma_k = \frac{c}{k+K}$, through simulations and some analyses, we found that:

If $E[||D_i||^2] \approx d$, the variance of the learning algorithm can be approximated by

$$Var_k \approx \frac{dc^2}{k+K}. \tag{58}$$

The detailed analysis about the variance are remains for the further exploration.

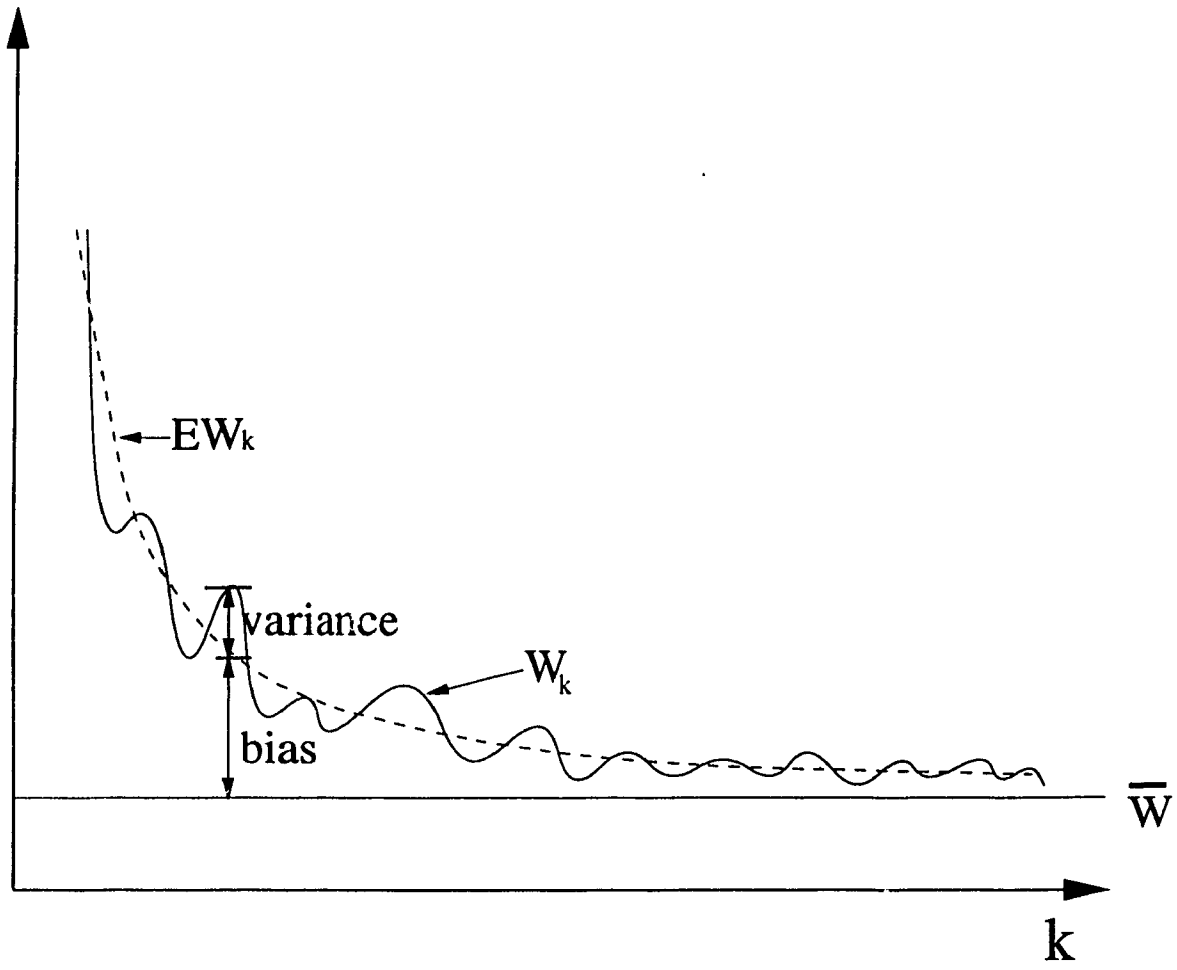

Figure 8: The relation of the learning error, the bias and the variance

2.4.1 Bias Analysis of Single Neuron Based Learning Algorithms

According to (46), we have

$$EW_{k+1} = EW_k + \gamma_k E(h(W_k)). \tag{59}$$

For single neuron based algorithms, $h()$ is a linear or a nearly linear vector function. So, $E(h(W_k)) \approx h(EW_k)$. Therefore, EW_k can be approximated by U_k under $U_0 = W_0$, where $\{U_k\}$ is a sequence defined by the DE (49).

If $\{W_k\}$ converges to $\overline{W}$. Then, $\{U_k\}$ is also convergent. Let $\overline{U} = \lim_{k\to\infty} U_k$. Thus, $\overline{U} = \overline{W}$.

According to the DE (49), we have

$$U_{k+1} - \overline{U} = -\sum_{i=k}^{\infty} \gamma_i h(U_i) \tag{60}$$

and

$$U_{k+1} - \overline{U} = U_k + \gamma_k h(U_k) - \overline{U}. \tag{61}$$

Since $h(\overline{U}) = 0$, (61) can be rewritten as

$$U_{k+1} - \overline{U} = U_k - \overline{U} + \gamma_k[h(U_k) - h(\overline{U})]. \tag{62}$$

For single neuron based learning algorithms, $h()$ is usually a linear or a nearly linear function. We have

$$h(U_k) - h(\overline{U}) \approx H(U_k - \overline{U}), \tag{63}$$

where $H = \frac{\partial h(U)}{\partial U}\Big|_{U=\overline{U}}$. Substituting (63) in (62), we obtain

$$U_{k+1} - \overline{U} \approx U_k - \overline{U} + \gamma_k H(U_k - \overline{U}) = (I + \gamma_k H)(U_k - \overline{U}), \tag{64}$$

where I is the identity matrix. Using the approximation (64) recursively, we get

$$U_{k+1} - \overline{U} \approx (U_0 - \overline{U}) \prod_{i=1}^{k} (I + \gamma_i H). \tag{65}$$

For single neuron based learning algorithms, H is a symmetric negative semidefinite matrix. If H is negative definite, its eigenvalues are all negative. Without loss of generality, let $H = diag(\lambda_{h1}, \lambda_{h2}, ..., \lambda_{hN})$, where $\lambda_{hN} \leq \lambda_{hN-1} \leq ... \leq \lambda_{h2} \leq \lambda_{h1} < 0$

For the jth component of U_{k+1}, equation (65) is written as

$$u_{j,k+1} - \overline{u}_j \approx (u_{j0} - \overline{u}_j) \prod_{i=1}^{k} (1 + \gamma_i \lambda_{jh}), \tag{66}$$

where $\overline{u}_j$ is the jth component of $\overline{U}$. Let $a_k = \prod_{i=1}^{k} (1 + \gamma_i \lambda_{jh})$ and take the logarithm of a_k to obtain

$$\ln a_k = \sum_{i=1}^{k} \ln(1 + \gamma_i \lambda_{jh}). \tag{67}$$

For small $\gamma_i \lambda_{jh}$, we have the following approximation:

$$\ln(1 + \gamma_i \lambda_{jh}) \approx \gamma_i \lambda_{jh}.$$

Replacing it in (67), we have

$$\ln a_k \approx \sum_{i=1}^{k} \gamma_i \lambda_{jh} = \lambda_{jh} \sum_{i=1}^{k} \gamma_i = \lambda_{jh} t_k, \tag{68}$$

where $t_k = \sum_{i=1}^{k} \gamma_i$. Thus $a_k \approx e^{\lambda_{jh} t_k}$. Substituting a_k in (66), we get

$$u_{j,k+1} - \overline{u}_j \approx (u_{j0} - \overline{u}_j) e^{\lambda_{jh} t_k}. \tag{69}$$

Therefore,

$$||U_{k+1} - \overline{U}|| = \sqrt{\sum_{i=1}^{N} (u_{j,k+1} - \overline{u}_j)^2} \approx \sqrt{\sum_{j=1}^{N} (u_{j0} - \overline{u}_j)^2 e^{2\lambda_{jh} t_k}}. \tag{70}$$

Since λ_{1h} is the largest eigenvalue, the error is mainly contributed by the first component of $U_{k+1} - \overline{U}$. The other components can be ignored. Hence,

$$||U_{k+1} - \overline{U}|| \approx |u_{10} - \overline{u}_1| e^{\lambda_{1h} t_k}. \tag{71}$$

If we let $\lambda_h = \lambda_{1h}$ and according to (57) and (71), we obtain the following approximation of the bias for single neuron based learning algorithms

$$Bias_k \approx |u_{10} - \overline{u}_1| e^{\lambda_{1h} t_k} = |w_{10} - \overline{w}_1| e^{\lambda_{1h} t_k} = |w_{10} - \overline{w}_1| e^{\lambda_h t_k}, \tag{72}$$

where w_{10} and $\overline{w}_1$ are the first components of initial weights W_0 and $\overline{W}$, respectively.

2.5 The Learning Rate Analysis

For single neuron based learning algorithms, the convergence speed of $Z(t)$ in the neighborhood of $\overline{Z}$ is determined by

$$H = \left. \frac{\partial h(Z)}{\partial Z} \right|_{Z=\overline{Z}}. \tag{73}$$

$||Z(t) - \overline{Z}|| = O(e^{\lambda_h t})$, where λ_h is the largest eigenvalue of H. The convergence speed for the solution to the DE is

$$||U_n - \overline{Z}|| = O(e^{\lambda_h t_n}). \tag{74}$$

Let $d \approx E(||D_\kappa||^2)$. The robustness can be approximated by

$$R \approx \frac{||\overline{Z}||^2}{d \sum_{k=1}^{\infty} \gamma_k^2}. \tag{75}$$

We see that the convergence speed depends on $t_n = \sum_{i=1}^{n-1} \gamma_i$ and the robustness depends on $r = \sum_{k=1}^{\infty} \gamma_k^2$. According to (153), $t_n \to \infty$ and $r < \infty$.

Let us assume that the learning rate has the form $\gamma_k = \frac{c}{(k+K)^s}$, where c is a constant and K is a large constant. The condition (153) is satisfied if $\frac{1}{2} < s \leq 1$. For this learning rate, we obtain

$$r = \sum_{k=1}^{\infty} \gamma_k^2 = \sum_{k=1}^{\infty} \frac{c^2}{(k+K)^{2s}} = c^2 \sum_{k=K+1}^{\infty} \frac{1}{k^{2s}} \approx (2s-1) K^{1-2s}.$$

Substituting this in (75), we have

$$R \approx \frac{||\bar{Z}||^2 K^{2s-1}}{c^2 d(2s-1)}. \tag{76}$$

Thus, for a small s, a large K should be used to have the same robustness.

For the learning rate $\gamma_k = \frac{c}{(k+K)^s}$, the convergence speed of single neuron based learning algorithms is different for the following three cases:

Case I: $s = 1$

$\gamma_k = \frac{c}{K+k}$ is the most commonly used learning rate. We obtain

$$t_n = \sum_{i=1}^{n-1} \gamma_i = \sum_{i=1}^{n-1} \frac{c}{K+i} = c(\ln(K+n) - \ln K) + O(1) = c \ln n + O(1).$$

According to (74), the convergence speed is

$$O(e^{\lambda_h t_n}) = O(e^{\lambda_h c \ln n}) = O(n^{c\lambda_h}). \tag{77}$$

Thus, single neuron based learning algorithms are polynomially convergent for the learning rate $\frac{c}{K+k}$.

Case II: $\frac{1}{2} < s < 1$

For $\frac{1}{2} < s < 1$, it is easy to verify

$$t_n = \sum_{i=1}^{n-1} \gamma_i = \sum_{i=1}^{n-1} \frac{c}{(K+i)^s} = \frac{c(1-s)}{(n+K)^{s-1}} + o(n^{1-s}) = \frac{c(1-s)}{n^{s-1}} + o(n^{1-s}).$$

According to (74), the convergence speed is obtained as

$$O(e^{\lambda_h t_n}) = O(e^{\lambda_h c(1-s) n^{1-s}}). \tag{78}$$

We see that single neuron based learning algorithms are exponentially convergent for the learning rate with $0 < s < 1$.

Case III: $s = 0, \gamma_k = c$

In this case, the learning rate is a constant. Then, the robustness of the learning algorithm is 0. The learning algorithm is not convergent in the sense of stochastic approximation.

The conclusion of Theorem 2.1 does not hold for this learning rate. However, U_k may be still convergent. The convergence speed of the learning algorithm in the neighborhood of $\overline{Z}$ can be obtained approximately by the following difference equation:

$$U_{k+1} = U_k + cHU_k, \tag{79}$$

where H is given in (73). Using (79) iteratively, we get

$$U_{k+1} = (I + cH)^k U_1.$$

Without loss of generality, let $H = diag(\lambda_h, \lambda_2, ..., \lambda_N)$, where $\lambda_h < 0$ is the largest eigenvalue of H. Hence, the convergence speed of the first component of U_{k+1} is

$$O((1 + c\lambda_h)^n),$$

which can be considered as the convergence speed of the learning algorithm. For a small $c\lambda_h$, we have $(1 + c\lambda_h)^n \approx e^{c\lambda_h n}$. Approximately, the convergence speed of the learning algorithm is

$$O(e^{c\lambda_h n}). \tag{80}$$

Thus, in this case, the algorithm is exponentially convergent and is faster than the algorithm with the learning rate of case II.

The convergence speed of $Z(t)$ is independent of the learning rate. Since

$$\sum_{k=1}^{\infty} \gamma_k^2 = \infty.$$

the assumption for Theorem 2.1 is not satisfied. Suppose that we can use Theorem 2.1, then $t_n = cn$ and the convergence speed of the learning algorithm is

$$O(e^{\lambda_h t_n}) = O(e^{c\lambda_h n}).$$

Hence, for a small $c\lambda_h$, Theorem 2.1 can provide an approximation of the convergence speed for a single neuron based learning algorithm with a constant learning rate.

2.5.1 Optimal Learning Rate Selection

We can see that the learning rate is one of the important parameters that determine the performance of learning algorithms. For a stochastic approximation learning algorithm, it is required that both the bias and the variance tend to 0 as $k \to \infty$.

For the learning rate γ_k, it is usually assumed that:

$$t_\infty = \sum_{i=1}^{\infty} \gamma_i = \infty \tag{81}$$

and

$$s_1 = \sum_{i=1}^{\infty} \gamma_i^2 < \infty. \tag{82}$$

If we choose the learning rate with the form $\gamma_k = \frac{c}{(K+k)^s}$, the conditions (81) and (82) are satisfied for $\frac{1}{2} < s \leq 1$.

For $s = 1$,

$$t_k = \sum_{i=1}^{k} \gamma_i = \sum_{i=1}^{k} \frac{c}{K+i} = c \sum_{i=1}^{k+K} \frac{1}{i} - c \sum_{i=1}^{K-1} \frac{1}{i} \approx c(\ln(k+K) - \ln K)), \tag{83}$$

and

$$s_k = \sum_{i=k}^{\infty} \gamma_i = \sum_{i=k}^{\infty} \frac{c^2}{(i+K)^2} \approx c^2 \frac{1}{k+K}.$$

According to (72), the bias of the learning algorithm is

$$Bias_k \approx |w_{10} - \overline{w}_1| e^{-\lambda_h t_k} \approx |w_{10} - \overline{w}_1| e^{-\lambda_h c(\ln(k+K) - \ln K)} = |w_{10} - \overline{w}_1| (\frac{k+K}{K})^{-c\lambda_h}. \quad (84)$$

Correspondingly, according to (58), the variance of the learning algorithm is

$$Var_k \approx dc^2 \frac{1}{k+K}. \quad (85)$$

The total learning error is

$$E[||\overline{W} - W_k||^2] = Bias_k^2 + Var_k \approx |w_{10} - \overline{w}_1|^2 (\frac{k+K}{K})^{2c\lambda_h} + dc^2 \frac{1}{k+K}. \quad (86)$$

Let $J(c) = |w_{10} - \overline{w}_1| (\frac{k+K}{K})^{c\lambda_h} + dc^2 \frac{1}{k+K}$. If all the parameters except c in (86) are fixed, we can find a c_o that minimizes $J(c)$. Then, the learning rate $\frac{c_o}{k+K}$ is a nearly optimal learning rate which minimizes the learning error $E[||\overline{W} - W_k||^2]$.

For the learning rate with the form $\frac{c}{(k+K)^s}$ ($\frac{1}{2} < s < 1$), it is not difficult to verify that the bias of single neuron based learning algorithms decreases exponentially. However, the variance usually decreases polynomially. There are no obvious differences between the learning performance with $s = 1$ and the learning performance with $s \neq 1$. Therefore, we can usually choose the learning rate with the form $\frac{c}{k+K}$.

2.6 Summary

In this chapter, the convergence speed, robustness, bias and variance of a neural network learning algorithm have been defined and analyzed. It is shown that the performances of single neuron based learning algorithms are mainly dependent on the learning rate and the eigenvalues of the learning samples. Based on the bias and variance analysis, an optimal learning rate can be selected. In the next chapter, we will present some detailed analysis and comparisons about the performances of the UP algorithms, the Oja algorithms and the Widrow-Hoff rule based algorithms.

Chapter 3

Analyses and Comparisons of Single Neuron Based Algorithms

3.1 Introduction

Using the definitions and the general analyses of stochastic approximation algorithms given in Chapter 2, we analyze the unsupervised Perceptron algorithms, Widrow-Hoff rule based algorithms and Oja learning algorithms in this chapter. The performance formulas of these learning algorithms are derived. Outliers are the noise samples or the samples with extreme values. In the presence of outliers, the robustness of learning algorithms can be largely reduced. We provide some theoretical and experimental analysis of outliers and show that the absolute criterion based learning algorithms are more robust than the square criterion based learning algorithms. In addition, some simulation results and comparisons are presented.

3.2 The NUP Algorithm Analysis

In Chapter 1, the normalized unsupervised Perceptron algorithm (NUP) was proposed. The norm of weight vector is always normalized as 1 in the NUP algorithm.

Like the UP algorithm, the NUP algorithm can also extract the first principal component of the input data. First, we analyze the convergence of the NUP algorithm.

3.2.1 Convergence of the NUP Algorithm

Substituting (18) in (19), we write the NUP learning equation as

$$W_{k+1} = \frac{W_k + \gamma_k Y_k}{||W_k + \gamma_k Y_k||}. \tag{87}$$

Using the expansion of $\frac{1}{1+x}$, we have

$$\frac{1}{||W_k + \gamma_k Y_k||} = 1 - \gamma_k ||Y_k|| + O(\gamma_k^2). \tag{88}$$

Substituting (88) in (87), we obtain

$$W_{k+1} = (W_k + \gamma_k Y_k)(1 - \gamma_k ||Y_k|| + O(\gamma_k^2)) = W_k + \gamma_k (Y_k - ||Y_k|| W_k) + \gamma_k A_k, \tag{89}$$

where $A_k = \gamma_k ||Y_k|| Y_k + O(\gamma_k) W_k + O(\gamma_k^2 Y_k)$ and $A_k \to 0$.

For the unsupervised Perceptrons, we have $||Y_k|| = ||X_k||$. Therefore,

$$E[(||Y_k||)|W_k] = E[(||X_k||)|W_k] = E(||X_k||) = E(||Y_k||)$$

and

$$E(||Y_k||^2|W_k) = E(||X_k||^2|W_k] = E(||X_k||^2) = E(||Y_k||^2).$$

We will simplify some representations of the conditional expectations in the following analysis.

Equation (89) can be rewritten as

$$W_{k+1} = W_k + \gamma_k [E(Y_k|W_k) - E(||Y_k||)W_k] + \gamma_k D_k + \gamma_k A_k \tag{90}$$

where $D_k = Y_k - E(Y_k|W_k) - ||Y_k||W_k + E(||Y_k||)W_k$.

It is obvious, $E(D_k|W_k) = 0$ and

$$E(||D_k||^2|W_k) \leq 4[E(||Y_k||^2) + ||E(Y_k|W_K)||^2 + ||W_k||^2 E(||Y_k||^2) + ||W_k||^2 (E||Y_k||)^2]$$

$$\leq 16E(||Y_k||^2) = 16E(||X_k||^2). \tag{91}$$

If $E(||X_k||^2)$ exists and condition (153) holds, the assumptions iii) and iv) in Theorem .1 are satisfied. Also, $E(||W_k||) = 1$.

According to (89), the criterion of the NUP algorithm can be considered as

$$J(W, X) = |WX| - ||X|| \frac{||W||^2}{2}$$

and

$$V(W) = E_X(|WX|) - E_X(||X||) \frac{||W||^2}{2}.$$

Using the above analysis and applying Corollary 2.1 directly, we have

Theorem 3.1 *Let S be the local maxima set of $V(W)$. Suppose that the attraction basin $D(S) = \mathbb{R}^N - \Omega$ (Ω is a null set), that $V(W)$ and its derivatives are continuous, and that the points in S are isolated. Assume that the learning rates satisfy (153), that $\{X_k\}$ is stationary and $E(||X_k||^2)$ is bounded. Then, W_k tends to a local maximum of $V(W)$ almost surely.*

For a general distribution, it is difficult to write $E_X(|WX|)$ as an explicit function of W. The properties of S are not easy to be verified. However, for normally distributed input data, we can obtain an explicit result. According to Lemma 1.1, for normally distributed samples with covariance C and mean zero vector, we have

$$E(Y_k|W_k) = \rho \frac{CW_k}{\sqrt{W_k^T C W_k}}, \tag{92}$$

where $\rho = \sqrt{\frac{2}{\pi}}$. It is easy to derive that $E(||Y_k||) = \rho\sqrt{\lambda_1}$. Substituting (92) in (90), we get the following LDE of the NUP algorithm:

$$W_{k+1} = W_k + \gamma_k\rho(\frac{CW_k}{\sqrt{W_k^T C W_k}} - \sqrt{\lambda_1}W_k) + \gamma_k D_k + \gamma_k A_k. \tag{93}$$

According to (3.2.1)

$$E(||D_k||^2|W_k) \leq 16E(||Y_k||^2) = 16E(||X_k||^2) = 16tr(C), \tag{94}$$

where $tr(C) = \sum_{i=1}^{N} \lambda_i$. Hence, $E(||D_k||^2)$ is bounded. For convenience, we treat $\rho\gamma_k$ as the learning rate of the LDE (93) and obtain the following ODE of the NUP algorithm:

$$\frac{dZ}{dt} = \frac{CZ}{\sqrt{Z^T C Z}} - \sqrt{\lambda_1}Z. \tag{95}$$

Let $h(Z) = \frac{CZ}{\sqrt{Z^T C Z}} - \sqrt{\lambda_1}Z$. The equilibrium state $\overline{Z}$ of (95) is the solution to $h(\overline{Z}) = 0$. That is

$$\frac{C\overline{Z}}{\sqrt{\overline{Z}^T C \overline{Z}}} - \sqrt{\lambda_1}\overline{Z} = 0. \tag{96}$$

It is easy to verify that $\overline{Z} = B_1$ and $-B_1$ are the solutions to (96), where B_1 is the first eigenvector corresponding to the largest eigenvalue λ_1 of C.

Let $V(Z) = \sqrt{Z^T C Z} - \sqrt{\lambda_1}\frac{Z^T Z}{2} - \frac{\sqrt{\lambda_1}}{2}$. Then,

$$V(Z) \leq \sqrt{\lambda_1}||Z|| - \sqrt{\lambda_1}\frac{||Z||^2}{2} - \frac{\sqrt{\lambda_1}}{2} = -\sqrt{\lambda_1}(||Z|| - 1)^2/2.$$

If $\lambda_1 > \lambda_2$, $V(Z)$ is negative definite. Since $\frac{\partial V(Z)}{\partial Z} = h(Z)$ and $\frac{dV(Z)}{dt} = h(Z)^T h(Z) \geq 0$, $V(Z)$ is a Liapunov function of (95) and $\overline{Z} = B_1$ and $-B_1$ are the only global maxima of $V(Z)$.

Also, $||W_k|| = 1$ is bounded and the conditions in Corollary 2.1 are satisfied. Consequently, the following theorem is proved:

Theorem 3.2 *For normally distributed stationary input samples, assume that the learning rate satisfies (153) and $\lambda_1 > \lambda_2$. Then, the weight vector trained by the NUP algorithm converges to the first unit eigenvector of C almost surely.*

3.2.2 Performance Analysis of the NUP Algorithm

We have proved the convergence of the NUP algorithm without solving its ODE and DE equations in the above analysis. To get the convergence speed of the NUP algorithm, we can solve its DE equation, or solve its ODE and apply Theorem 2.1. The simplest approach is to analyze the matrix $H = \frac{\partial h(Z)}{\partial Z}\Big|_{Z=\bar{Z}}$. For normally distributed samples, according to (95), we have

$$H = \frac{\partial h(Z)}{\partial Z}\Big|_{Z=\bar{Z}} = \frac{1}{\sqrt{\lambda_1}}C - \sqrt{\lambda_1}I - \sqrt{\lambda_1}B_1B_1^T$$

where I is the identity matrix. Applying the orthogonal transform matrix B to H, we have

$$B^THB = \frac{1}{\sqrt{\lambda_1}}B^TCB - \sqrt{\lambda_1}I - \sqrt{\lambda_1}B^TB_1B_1^TB = diag(-\sqrt{\lambda_1}, \frac{\lambda_2-\lambda_1}{\sqrt{\lambda_1}}, ..., \frac{\lambda_N-\lambda_1}{\sqrt{\lambda_1}}),$$

$\lambda_h = \frac{\lambda_2-\lambda_1}{\sqrt{\lambda_1}}$ is the largest eigenvalue of H. Hence, in the neighborhood of $\bar{Z}$, the convergence speed for the ODE (95) of the NUP algorithm is $O(e^{\frac{\lambda_2-\lambda_1}{\sqrt{\lambda_1}}t})$.

According to (77), the learning speed of the NUP algorithm is $O(n^{c\rho\frac{\lambda_2-\lambda_1}{\sqrt{\lambda_1}}})$ for the learning rate $\gamma_n = \frac{c}{n+K}$.

By directly solving the DE or ODE equations of the NUP algorithm, we can obtain the same convergence speed.

Then, we calculate the robustness of the NUP algorithm. For the GUP algorithm,

$$D_k = Y_k - E(Y_k|W_k) - ||Y_k||W_k + E(||Y_k||)W_k.$$

The robustness is mainly used for analyzing the qualitative properties of a learning algorithm. The exact estimation of the variance of D_k is difficult to derive and is

also not important for robustness analysis. We only give a rough approximation here. Since the first term of D_k contributes most to its variance, roughly, we have,

$$E(||D_k||^2|W_k) \approx E(||Y_k||^2) = tr(C).$$

According to (76), the robustness of the NUP algorithm for the learning rate $\gamma_n = \frac{c}{n+K}$ is

$$R_{NUP} = \frac{||\bar{Z}||^2}{\sum_{k=1}^{\infty} \gamma_k^2 E(||D_k||^2)} \approx \frac{1}{c^2 tr(C) \sum_{k=1}^{\infty} \gamma_k^2} \approx \frac{K}{c^2 tr(C)}.$$

Substituting λ_h in (84), we get the bias of the NUP learning algorithm

$$Bias_{NUP} \approx |w_{20} - \overline{w}_2|(\frac{k+K}{K})^{c\rho\lambda_h} = |w_{20}|(\frac{k+K}{K})^{\frac{c\rho(\lambda_2-\lambda_1)}{\sqrt{\lambda_1}}}$$

$$= |W_0^T B_2|(\frac{k+K}{K})^{\frac{c\rho(\lambda_2-\lambda_1)}{\sqrt{\lambda_1}}}, \quad (97)$$

where $||W_0|| = 1$.

3.3 GUP Learning Algorithm Analysis

Similar to the analysis of the NUP algorithm, the analysis of the GUP learning algorithm is presented in this section.

3.3.1 Convergence of the GUP Algorithm

The learning equation of the GUP algorithms is

$$W_{k+1} = W_k + \gamma_k(Y_k - W_k). \quad (98)$$

The criterion function of the GUP algorithm can be considered as

$$J(W, X) = |WX| - \frac{||W||^2}{2},$$

and

$$V(W) = E_X(|WX|) - \frac{||W||^2}{2}.$$

According to Lemma 1.1, for normally distributed samples with covariance C and mean zero vector, we have

$$E(Y_k|W_k) = \rho \frac{CW_k}{\sqrt{W_k^T C W_k}}. \tag{99}$$

Substituting (99) in (98), we can write (98) as

$$W_{k+1} = W_k + \gamma_k(\rho \frac{CW_k}{\sqrt{W_k^T C W_k}} - W_k) + \gamma_k D_k. \tag{100}$$

Comparing (98) and (100), we obtain

$$D_k = Y_k - \rho \frac{CW_k}{\sqrt{W_k^T C W_k}}.$$

Hence,

$$E(||D_k||^2|W_k) = E(D_k^T D_k|W_k) < 2E(||Y_k||^2) + 2\rho^2 E(\frac{W_k^T C^T C W_k}{W_k^T C W_k}) \leq 2\sum_{m=1}^{N} \lambda_m + 2\rho^2\lambda_1.$$

Thus, $E(||D_k||^2)$ is bounded. The ODE of the GUP algorithm is

$$\frac{dZ}{dt} = \rho \frac{CZ}{\sqrt{Z^T C Z}} - Z. \tag{101}$$

Let $h(Z) = \rho \frac{CZ}{\sqrt{Z^T C Z}} - Z$. It is easy to verify that $\overline{Z} = \rho\sqrt{\lambda_1} B_1$ or $-\rho\sqrt{\lambda_1} B_1$ is the solution to the equation $h(Z) = 0$, where B_1 is the first eigenvector corresponding to the largest eigenvalue λ_1 of C.

Let $V(Z) = \rho\sqrt{Z^T C Z} - \frac{Z^T Z}{2} - \frac{\rho^2\lambda_1}{2}$. Then,

$$V(Z) \leq \rho\sqrt{\lambda_1}||Z|| - \frac{||Z||^2}{2} - \frac{\rho^2\lambda_1}{2} = -(||Z|| - \rho\sqrt{\lambda_1})^2/2.$$

If $\lambda_1 > \lambda_2$. $V(Z)$ is negative definite. Since $\frac{\partial V(Z)}{\partial Z} = h(Z)$ and $\frac{dV(Z)}{dt} \geq 0$, $V(Z)$ is a Liapunov function of (101). It is not difficult to show that $\overline{Z}$ and $-\overline{Z}$ are the only global maxima of $V(Z)$ and that $E(||W_k||)$ is bounded.

According to Corollary 2.1, the following theorem is proved:

Theorem 3.3 *For normally distributed stationary input samples, assume that the learning rate satisfies (153) and $\lambda_1 > \lambda_2$. Then, the weight vector trained by the GUP algorithm converges to $\rho\sqrt{\lambda_1}B_1$ or $-\rho\sqrt{\lambda_1}B_1$ almost surely.*

3.3.2 Performance Analysis of the GUP Algorithm

For normally distributed samples, according to (101), we have

$$H = \left.\frac{\partial h(Z)}{\partial Z}\right|_{Z=\bar{Z}} = \frac{1}{\lambda_1}C - I - B_1 B_1^T.$$

Apply the orthogonal transform matrix B to H,

$$B^T H B = diag(-1, \frac{\lambda_2 - \lambda_1}{\lambda_1}, ..., \frac{\lambda_N - \lambda_1}{\lambda_1}).$$

Thus, $\lambda_h = \frac{\lambda_2 - \lambda_1}{\lambda_1}$ is the largest eigenvalue of H. The convergence speed for the ODE (101) of the GUP algorithm is $O(e^{\frac{\lambda_2 - \lambda_1}{\sqrt{\lambda_1}}t})$.

According to (77), the learning speed of the GUP algorithm is $O(n^{-\frac{\lambda_2 - \lambda_1}{\lambda_1}})$ for the learning rate $\gamma_k = \frac{c}{k+K}$. The UP algorithm is a special GUP algorithm with learning rate $\gamma_k = \frac{1}{k}$. The convergence speed of the UP algorithm is $O(n^{-\frac{\lambda_2 - \lambda_1}{\lambda_1}})$, which is the same as the convergence speed (35) which is directly derived from the DE of the UP algorithm in Chapter 1.

For the GUP algorithm, since $D_k = Y_k - \rho\frac{CW_k}{\sqrt{W_k^T C W_k}}$,

$$d = E(||D_k||^2 | W_k) = E(||Y_k||^2) - \rho^2 E(\frac{W_k^T C^T C W_k}{W_k^T C W_k})$$

$$\approx \sum_{m=1}^{N} \lambda_m - \rho^2\lambda_1 = tr(C) - \rho\lambda_1 \approx tr(C).$$

According to (76), the robustness of the GUP algorithm for the learning rate $\gamma_n = \frac{c}{n+k}$ is

$$R_{NUP} = \frac{||\bar{Z}||^2}{\sum_{k=1}^{\infty} \gamma_k^2 E(||D_k||^2)} \approx \frac{\rho^2 \lambda_1}{c^2 tr(C) \sum_{k=1}^{\infty} \gamma_k^2} \approx \frac{K\rho^2 \lambda_1}{c^2 tr(C)}.$$

Substituting λ_k in (84), we get the bias of the GUP learning algorithm

$$Bias_{GUP} \approx (w_{20} - \overline{w}_2)(\frac{k+K}{K})^{c\lambda_k} = |w_{20}|(\frac{k+K}{K})^{\frac{c(\lambda_2-\lambda_1)}{\lambda_1}}. \tag{102}$$

In (36), we have a more accurate approximation of the solution to the ODE of the UP algorithm:

$$y_i(t) \approx y_i(0)(\frac{\rho\sqrt{\lambda_1}}{y_1(0)})^{\frac{\lambda_i}{\lambda_1}} e^{(\eta_i - 1)t} = B_i^T Z(0)(\frac{\rho\sqrt{\lambda_1}}{B_1^T Z(0)})^{\frac{\lambda_i}{\lambda_1}} e^{(\eta_i - 1)t}. \tag{103}$$

Therefore, the bias of the GUP algorithm can be approximated by

$$Bias_{GUP} \approx B_i^T Z(0)(\frac{\rho\sqrt{\lambda_1}}{B_1^T Z(0)})^{\frac{\lambda_i}{\lambda_1}}(\frac{k+K}{K})^{\frac{c(\lambda_2-\lambda_1)}{\lambda_1}} = \tilde{w}_0(\frac{k+K}{K})^{\frac{c(\lambda_2-\lambda_1)}{\lambda_1}}, \tag{104}$$

where $\tilde{w}_0 = B_i^T Z(0)(\frac{\rho\sqrt{\lambda_1}}{B_1^T Z(0)})^{\frac{\lambda_i}{\lambda_1}}$. According to (85), the variance of the GUP learning algorithm is

$$Var_{GUP} \approx \frac{c^2 d}{k+K} \approx \frac{c^2 tr(C)}{k+K}. \tag{105}$$

Substituting (105) and (104) in (86), we have the total learning error of the GUP learning algorithm

$$E[||\hat{W} - W_k||^2] = Bias_{GUP}^2 + Var_{GUP} \approx (W_0^T B_2)^2(\frac{k+K}{K})^{\frac{2c(\lambda_2-\lambda_1)}{\lambda_1}} + \frac{c^2 tr(C)}{k+K}. \tag{106}$$

For principal component analysis, the projection of W_k at the second principal component B_2 contributes the main error of the leaning algorithm. It is not difficult

to derive that the mean squared error(MSE) of the GUP algorithm at the second principal component is

$$MSE_{GUP} = E(|W_k^T B_2|^2) \approx \tilde{w}_0^2 (\frac{k+K}{K})^{\frac{2c(\lambda_2-\lambda_1)}{\lambda_1}} + \frac{c^2\lambda_2}{k+K}. \quad (107)$$

3.3.3 Simulation Results of the GUP Algorithm

For $\lambda_1 = 1, \lambda_2 = 0.9, \tilde{w}_0 = 1$, $k = 10000, K = 2000$, Fig. 9 shows that $\sqrt{MSE_{GUP}}$ has the minimal value at $c = 14$. Fig. 10, Fig. 11 and Fig. 12 illustrate the learning results for $c = 5$, $c = 14$, and $c = 25$, respectively. We see that the learning algorithm has large bias for $c = 5$ and the variance is too large for $c = 25$. For $c = 14$, the learning error is smallest among the three learning cases.

3.4 Widrow-Hoff Rule Based Algorithm Analysis

The criterion of the *least-mean-square rule (LMS)* or *the delta rule* or the *Widrow-Hoff rule* can be conveniently employed to derive the following training algorithm. The neural network corresponding to this rule is a single linear neuron. Let X represent the input pattern and y is the desired output for X. Consider the criterion

$$J(W, X) = -\frac{1}{2}(y - W^T X)^2.$$

Taking the partial derivative of J with respect to W yields

$$\frac{\partial J}{\partial W} = (y - W^T X)X.$$

That is $\tilde{h}(W_k, X_k) = (y_k - W_k^T X_k)X_k$. Substituting it in the general algorithm of (45), we obtain the following LDE of the Widrow-Hoff algorithm:

$$W_{k+1} = W_k + \gamma_k(y_k - W_k^T X_k)X_k. \quad (108)$$

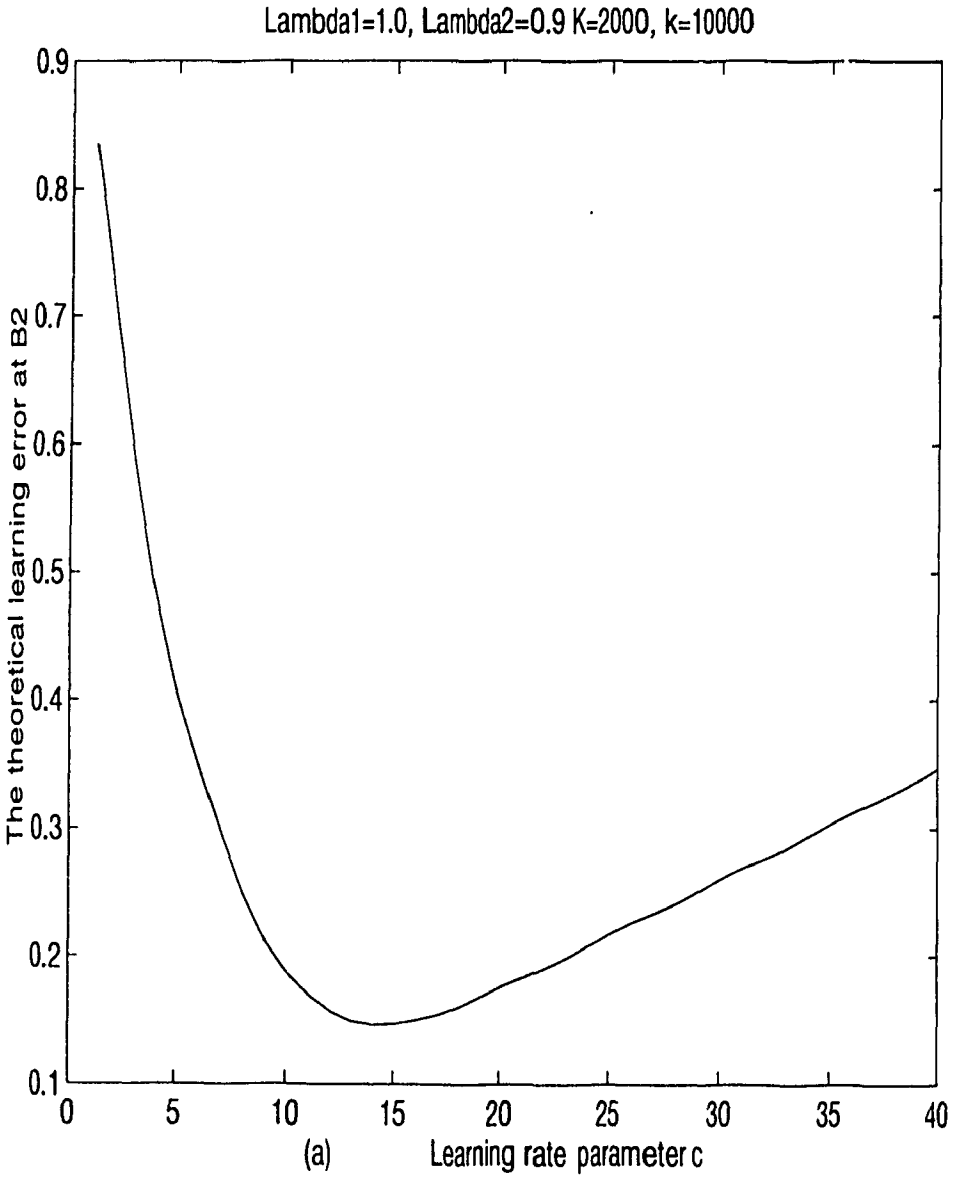

Figure 9: The optimal learning rate of the GUP algorithm

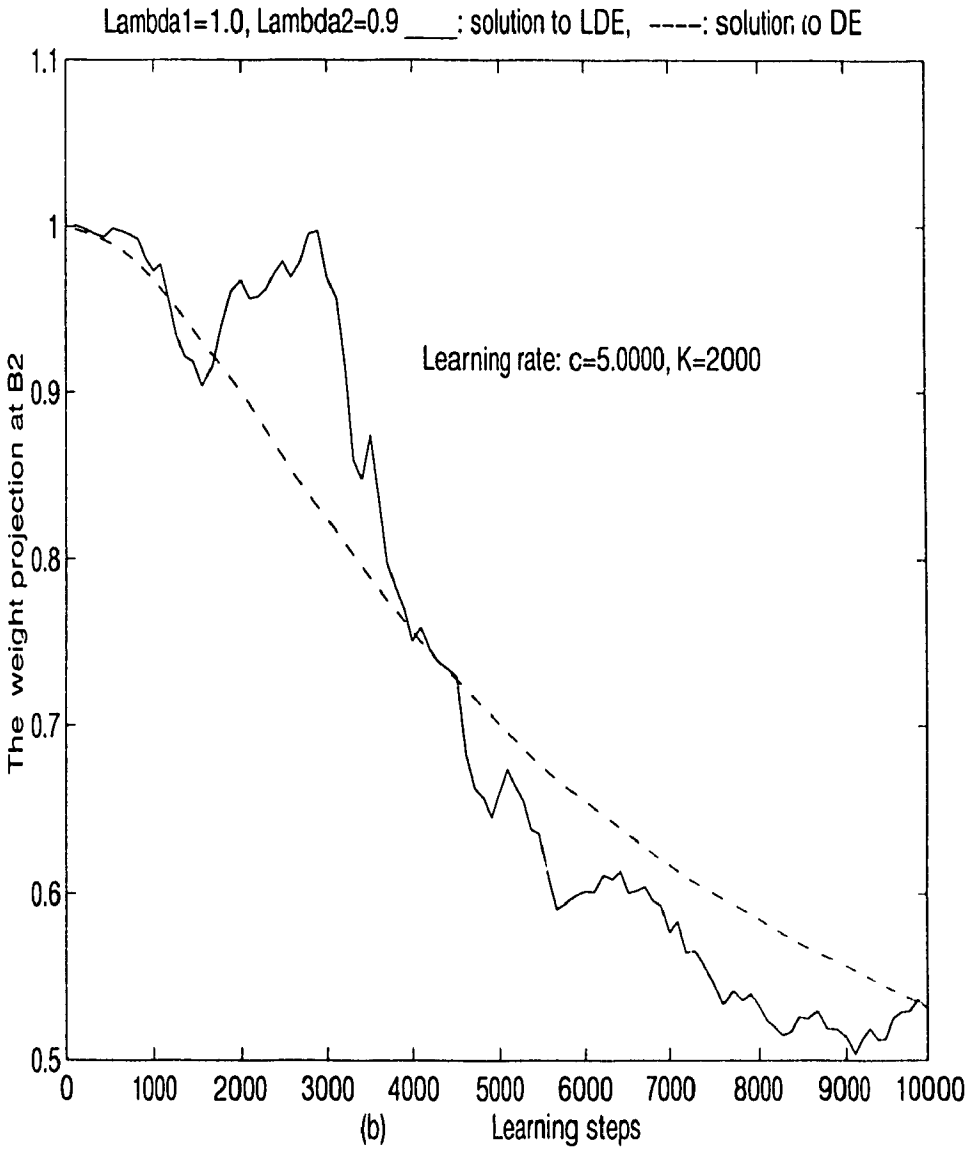

Figure 10: The learning results of the GUP algorithm with $c = 5$

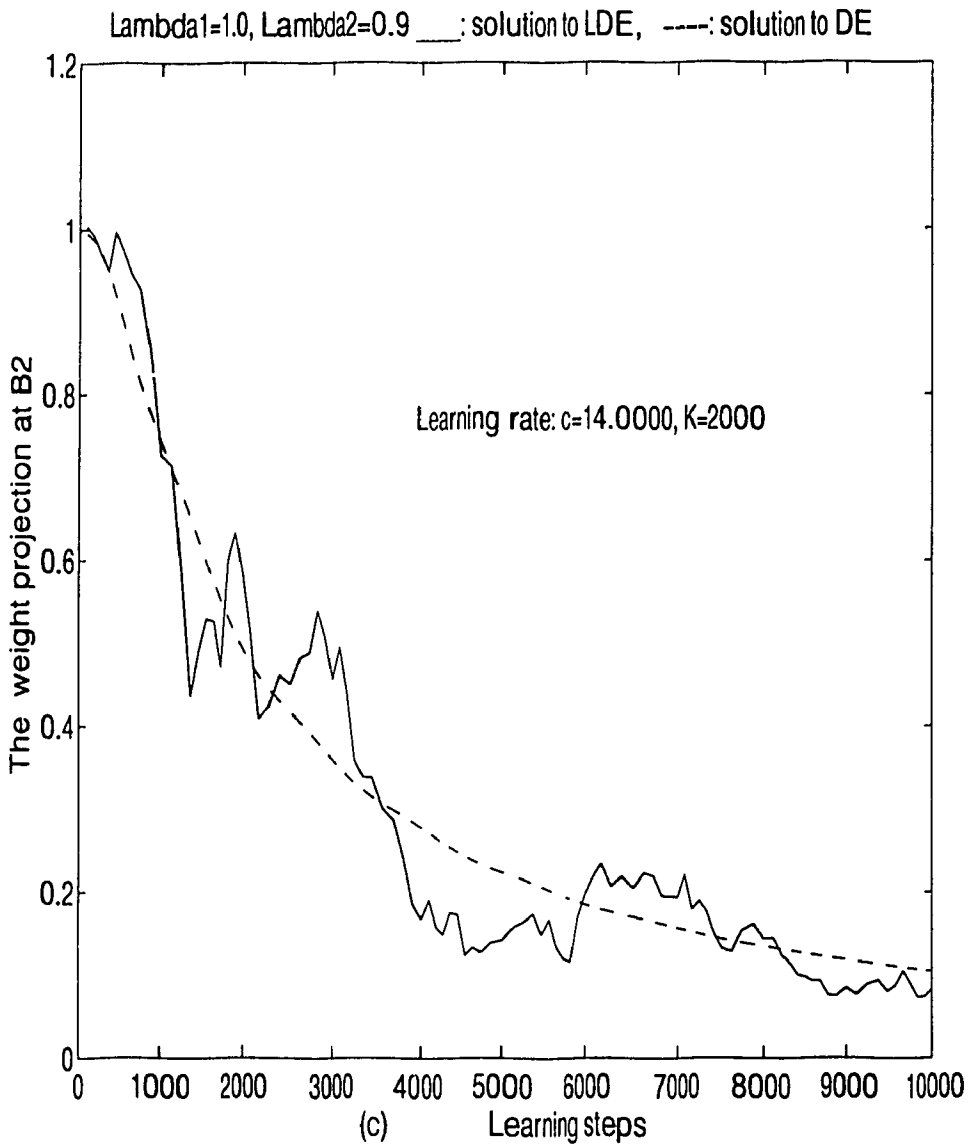

Figure 11: The learning results of the GUP algorithm with $c = 14$

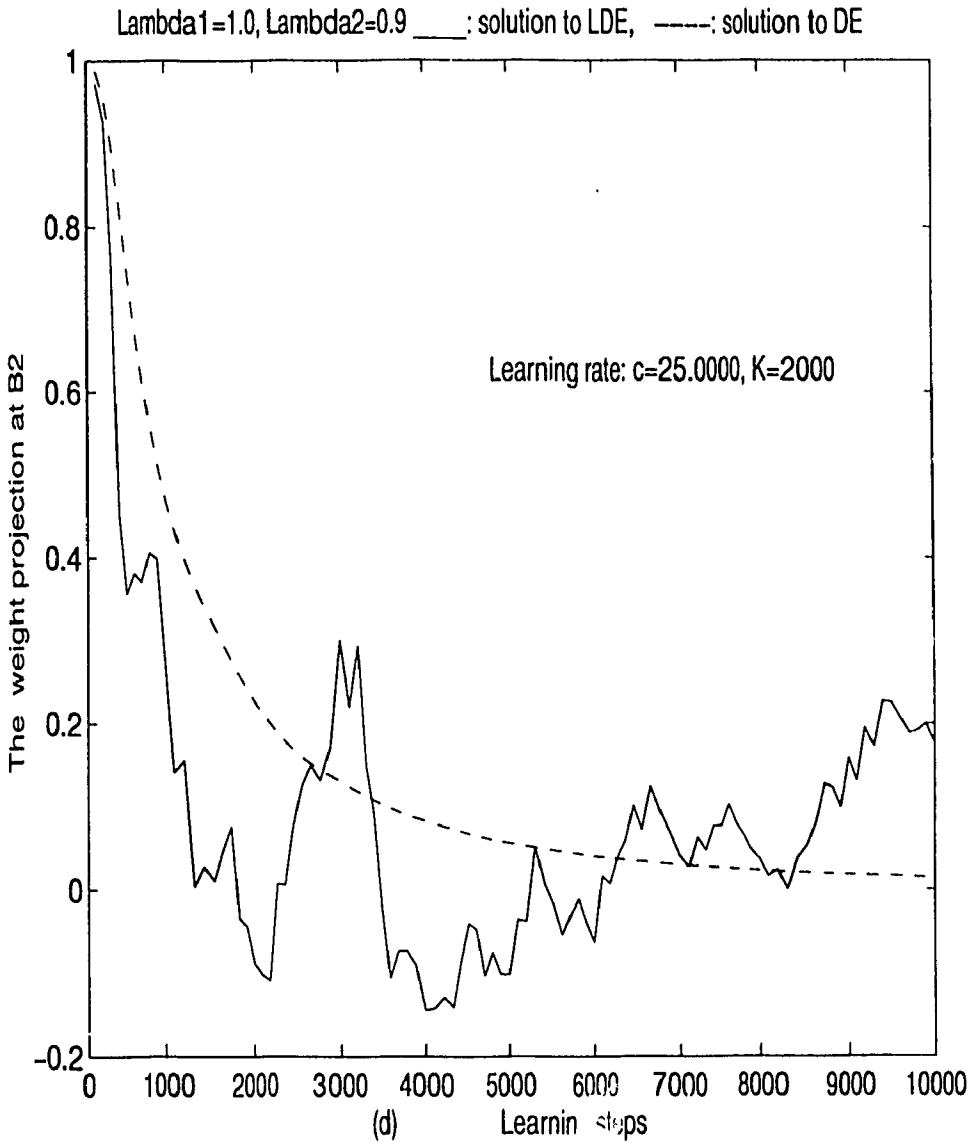

Figure 12: The learning results of the GUP algorithm with $c = 25$

3.5 Convergence of the Widrow-Hoff Algorithm

For the constant learning rate, the convergence of the Widrow-Hoff algorithm was introduced in Chapter 0. In this section, we focus on the convergence analysis of the Widrow-Hoff algorithm with a stochastic approximation learning rate.

Let $V(W) = E_X(J(W, X))$, which is

$$V(W) = -\frac{1}{2}(E(y^2) - 2W^T E(yX) + W^T E(XX^T)W) = -\frac{1}{2}(e - 2W^T R + W^T CW), \tag{109}$$

where $e = E(y^2), R = E(yX)$ and $C = E(XX^T)$ which is positive semidefinite. Hence, $h(W) = \frac{\partial V}{\partial W} = R - CW$. Substitute it in (47). Thus, the ODE of the Widrow-Hoff algorithm is

$$\frac{dZ}{dt} = R - CZ, \tag{110}$$

which is a linear system. Assume that C is nonsingular. Then, $V(W)$ is negative definite. There is only one asymptotically stable state $\overline{Z} = C^{-1}R$ to (110).

$$||Z(t) - \overline{Z}|| = O(e^{-\lambda_{min}t}),$$

where λ_{min} is the smallest eigenvalue of C.

Let $D_k = \tilde{h}(W_k, X_k) - h(W_k)$, we have $E(D_k) = 0$ and

$$E(||D_k||^2|W_k) = E(||\tilde{h}(W_k, X_k) - h(W_k)||^2|W_k) = E(||\tilde{h}(W_k, X_k)||^2|W_k) - ||h(W_l||^2$$

$$\leq E(||\tilde{h}(W_k, X_k)||^2|W_k) = E(||-(y_k - W_k^T X_k)X_k||^2|W_k)$$

$$\leq 2E(y_k^2||X_k||^2) + 2W_k^T E(X_k X_k^T X_k X_k^T)W_k.$$

Thus, if $E(y^2||X||^2)$ and $E(XX^TXX^T)$ exist, $E(||D_k||^2)$ is bounded. Consequently, the following theorem is obtained:

Theorem 3.4 *For stationary input data, assume that i) $E(y^2||X||^2)$ and $E(XX^TXX^T)$ exist, ii) the learning rates satisfy (153) and iii) C is nonsingular and $E(||W_k||)$ is bounded. Then, the weight vector trained by the Widrow-Hoff rule (108) converges to $C^{-1}R$ almost surely, and the convergence speed is $O(n^{-c\lambda_{min}})$.*

Remarks: Note that $E(y^2||X||)^2 \leq \frac{1}{2}(E(y^4) + E(||X||^4))$ and

$$||E(XX^TXX^T)|| \leq E(||X||^4).$$

If $E(y^4)$ and $E(||X||^4)$ exist, the condition i) in Theorem 3.4 is also satisfied. For the pattern classification problem, $y = 1$ or -1, the condition i) only requires that $E(XX^TXX^T)$ exists. A similar convergence theorem for classification problems was given in [4].

To get the robustness of the Widrow-Hoff learning algorithm, we need to estimate

$$E(||D_k||^2|W_k) = E(||\tilde{h}(W_k, X_k) - h(W_k)||^2) \approx E(||\tilde{h}(W_k, X_k)||^2)$$

$$\approx E(||-(y_k - W_k^TX_k)X_k||^2) \approx E(||y_kX_k||^2) + W_k^TE(X_kX_k^TX_kX_k^T)W_k$$

$$\approx ||R||^2 + \bar{Z}^TE(X^TX)E(XX^T)\bar{Z} \approx ||R||^2 + ||\bar{Z}||^2tr(C)\lambda_1.$$

Thus, according to (76), the robustness of the LMS learning algorithm can be defined as

$$R_{LMS} = \frac{K||\bar{Z}||^2}{c^2(||R||^2 + ||\bar{Z}||^2tr(C)\lambda_1)}.$$

According to (84), the bias of the Widrow-Hoff learning algorithm is

$$Bias_{LMS} \approx B_{min}^T(W_0 - \bar{Z})(\frac{k+K}{K})^{-c\lambda_{min}}, \tag{111}$$

where B_{min} is the eigenvector of C corresponding to the eigenvalue λ_{min}.

3.6 Oja Learning Algorithm Analysis

In 1982, Oja demonstrated that a single linear neuron with a Hebbian-type adaptation rule for its synaptic weights can evolve into a filter for the first principal component of the input distribution[53]. Specifically, the Oja's SGA learning equation is of the form

$$\tilde{W}_{k+1} = W_k + \gamma_k X_k X_k^T W_k, \tag{112}$$

$$W_{k+1} = \frac{\tilde{W}_{k+1}}{||\tilde{W}_{k+1}||}, \tag{113}$$

where X_k is input vector and W_k is the weight vector. Under certain conditions, Oja proved that W_k tends almost surely to the first unit eigenvector of $C = E(X_k X_k^T)$ corresponding to the largest eigenvalue.

It is not difficult to derive that the ODE of the SGA algorithm is

$$\frac{dZ}{dt} = CZ - ||CZ||Z. \tag{114}$$

Equation (114) has only two asymptotically stable states $\overline{Z} = B_1$ or $\overline{Z} = -B_1$. The local asymptotical behaviors of the SGA algorithm is described by

$$H = \left.\frac{\partial h(Z)}{\partial Z}\right|_{Z=\overline{Z}} = C - ||CB_1||I - \frac{CCB_1B_1^T}{||CB_1||} = C - \lambda_1 I - \lambda_1 B_1 B_1^T.$$

Applying the orthogonal transform B, we obtain

$$B^T H B = B^T C B - \lambda_1 I - \lambda_1 B^T B_1 B_1^T B = diag(-\lambda_1, \lambda_2 - \lambda_1, ..., \lambda_N - \lambda_1).$$

We see that $\lambda_h = \lambda_2 - \lambda_1$ is the largest eigenvector of H. Thus, for the learning rate $\frac{c}{n+K}$, the learning speed of the SGA algorithm is $O(n^{c(\lambda_2-\lambda_1)})$. This result can also be derived by directly solving the ODE or DE of the SGA algorithm.

For the SGA algorithm,

$$D_k = X_k X_k^T W_k - ||X_k X_k^T W_k|| W_k - (CW_k - ||CW_k|| W_k).$$

The robustness is mainly used for analyzing the qualitative properties of a learning algorithm. We can use some approximation to compute it. Since the first term of D_k contributes the most to the variance of D_k, approximately, we have

$$E(||D_k||^2|W_k) \approx W_k^T E(X_k X_k^T X_k X_k^T) W_k = W_k^T E(X_k^T X_k X_k X_k^T) W_k$$

$$\approx W_k^T E(X_k^T X_k) E(X_k X_k^T) W_k = W_k^T C tr(C) W_k \approx \lambda_1 tr(C)$$

According to (76), for the learning rate $\frac{c}{k+K}$, the robustness of the SGA algorithm is

$$R_{Oja} = \frac{K}{c^2 tr(C)\lambda_1}.$$

Substituting λ_k in (84), the bias of the Oja's SGA learning algorithm is

$$Bias_{Oja} \approx B_2^T W_0 (\frac{k+K}{K})^{c(\lambda_2 - \lambda_1)},$$

where $||W_0|| = 1$.

Also, Oja and Karhunen gave the following the linearized SGA algorithm to extract principal components:

$$W_{k+1} = W_k + \gamma_k [X_k X_k^T W_k - (W_k^T X_k X_k^T W_k) W_k]. \tag{115}$$

When W_k is a vector, Oja proved that W_k converges to the first principal component[55].

In the matrix case, the asymptotical properties of the ordinary differential equation corresponding to (115) were discussed in [84].

The ODE corresponding to the learning algorithm (115) is

$$\frac{dZ}{dt} = CZ - Z^T CZZ. \tag{116}$$

Equation (116) has only one asymptotically stable state $\bar{Z} = B_1$ or $\bar{Z} = -B_1$. To analyze the learning speed of the linearized SGA algorithm, we have

$$H = \left.\frac{\partial h(Z)}{\partial Z}\right|_{Z=\bar{Z}} = C - B_1^T C B_1 I - 2CB_1 B_1 = C - \lambda_1 I - 2\lambda_1 B_1 B_1.$$

Similar to the analysis of the SGA algorithm, it can be derived that the learning speed of the linearized SGA algorithm is also $O(n^{c(\lambda_2-\lambda_1)})$, and the robustness and the bias of the linearized SGA are the same as that of the SGA algorithm.

3.6.1 Comparison of Neural Network PCA Algorithms

The above analysis for single neuron based learning algorithms can be summarized in the Table 2.

Algorithms	GUP	NUP	Oja	Widrow-Hoff
Learning Type	Unsupervised	Unsupervised	Unsupervised	Supervised
Criterion	Quadratic	Absolute	Absolute	Quadratic
Conv. Speed	$O(n^{c\frac{\lambda_2-\lambda_1}{\lambda_1}})$	$O(n^{c\rho\frac{\lambda_2-\lambda_1}{\sqrt{\lambda_1}}})$	$O(n^{c(\lambda_2-\lambda_1)})$	$O(n^{-c\lambda_{min}})$
Robustness	$\frac{\rho^2\lambda_1 K}{c^2 tr(C)}$	$\frac{K}{c^2 tr(C)}$	$\frac{K}{c^2 tr(C)\lambda_1}$	$\frac{\|\bar{Z}\|^2 K}{c^2(\|R\|^2+\|\bar{Z}\|^2 tr(C)\lambda_1)}$
Bias	$\tilde{w}_0(\frac{k+K}{k})^{\frac{c(\lambda_2-\lambda_1)}{\lambda_1}}$	$\tilde{w}_0(\frac{k+K}{K})^{\frac{c\rho(\lambda_2-\lambda_1)}{\sqrt{\lambda_1}}}$	$\tilde{w}_0(\frac{k+K}{k})^{c(\lambda_2-\lambda_1)})$	$\tilde{w}_0(\frac{k+K}{K})^{-c\lambda_{min}}$

Table 2: The performance of single neuron based learning algorithms

$\tilde{w}_0$ in Table 2 is the initial bias of the learning algorithms. According to Table 2, theoretically, the learning speed of a learning algorithm can be designed as fast as possible by choosing large enough learning rate parameters or scaling the training data. However, the robustness of the learning algorithm will decrease with the corresponding rate. Hence, the convergence speed and the robustness must be analyzed together.

Assume that the learning rate for the NUP algorithm is $\frac{c}{k+K}$ and let $c' = c\rho\sqrt{\lambda_1}$. With the learning rate $\frac{c'}{k+K}$, the convergence speed of the GUP algorithm is

$$O(n^{c'\frac{\lambda_2-\lambda_1}{\lambda_1}}) = O(n^{c\rho\frac{\lambda_2-\lambda_1}{\sqrt{\lambda_1}}})$$

and the robustness of the GUP algorithm is

$$\frac{\rho^2\lambda_1 K}{(c')^2 tr(C)} = \frac{\rho^2\lambda_1 K}{c^2\rho^2\lambda_1 tr(C)} = \frac{K}{c^2 tr(C)}.$$

We see that the two learning algorithms have the same learning speeds and robustness with a proper choice of learning rates. Therefore, the NUP and the GUP algorithms are equivalent

Also, we can compare the unsupervised Perceptron algorithms with the Oja learning algorithms. Let $c' = \frac{c}{\lambda_1}$, the learning speed of the Oja learning algorithms is the same as the learning speed of the GUP algorithm, and the robustness of the Oja learning algorithms is

$$\frac{K}{(c')^2 tr(C)\lambda_1} = \frac{K}{\frac{c^2}{\lambda_1^2} tr(C)\lambda_1} = \frac{\lambda_1 K}{c^2 tr(C)}$$

which is slightly larger than the robustness of the GUP algorithm $\frac{\lambda_1\rho}{c^2 tr(C)}$. In other words, if we design the two learning algorithms with the same robustness, the Oja learning algorithms converge slightly faster than the unsupervised Perceptron algorithms. Fig. 13 shows the convergence speed for the DE equations of the Oja algorithm and the GUP algorithm under the same environment. Fig. 14 demonstrates the learning results and the robustness of the two algorithms. We can see that there are no obvious differences between performances of the two algorithms.

In the next section, we will show that the unsupervised Perceptron algorithms are much more robust than the Oja learning algorithms in the presence of outliers.

3.7 Outlier Analysis of Linear Learning Algorithms

An outlier is an extreme value of a variable which can be both input and output vectors. Outliers can significantly deteriorate the performances of a learning algorithm.

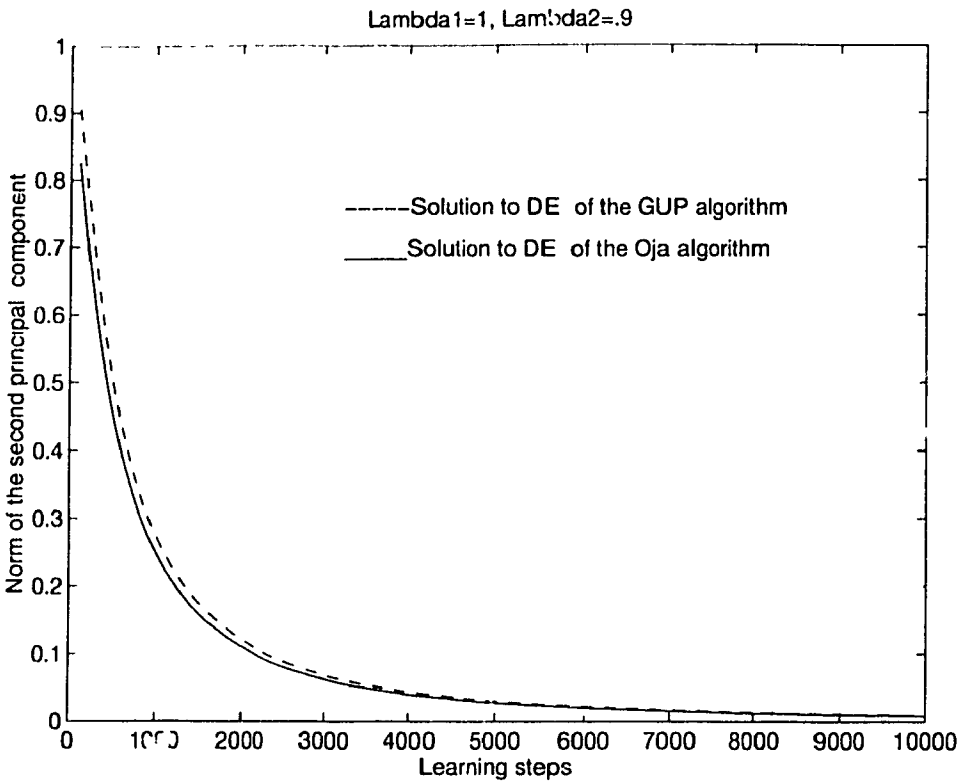

Figure 13: The learning speed of the GUP and Oja algorithms

In the literature of statistics, a number of techniques have been developed to tackle the problem[13, 14, 33]. Recently, more attention is being paid to the importance of this problem in neural network learning. In [82, 35], the robust PCA learning approaches are discussed.

Based on our definitions and analyses for single neuron based learning algorithms, in this section, we theoretically analyze how outliers deteriorate the performances of a learning algorithm.

3.7.1 Outlier Analysis of Unsupervised Learning Algorithms

For unsupervised learning algorithms, only input vectors contain outliers. Assume that the outliers appear with a frequency p for a learning algorithm. Let X, O and Y represent normal input vector, outlier vector, and actual input vector, respectively.

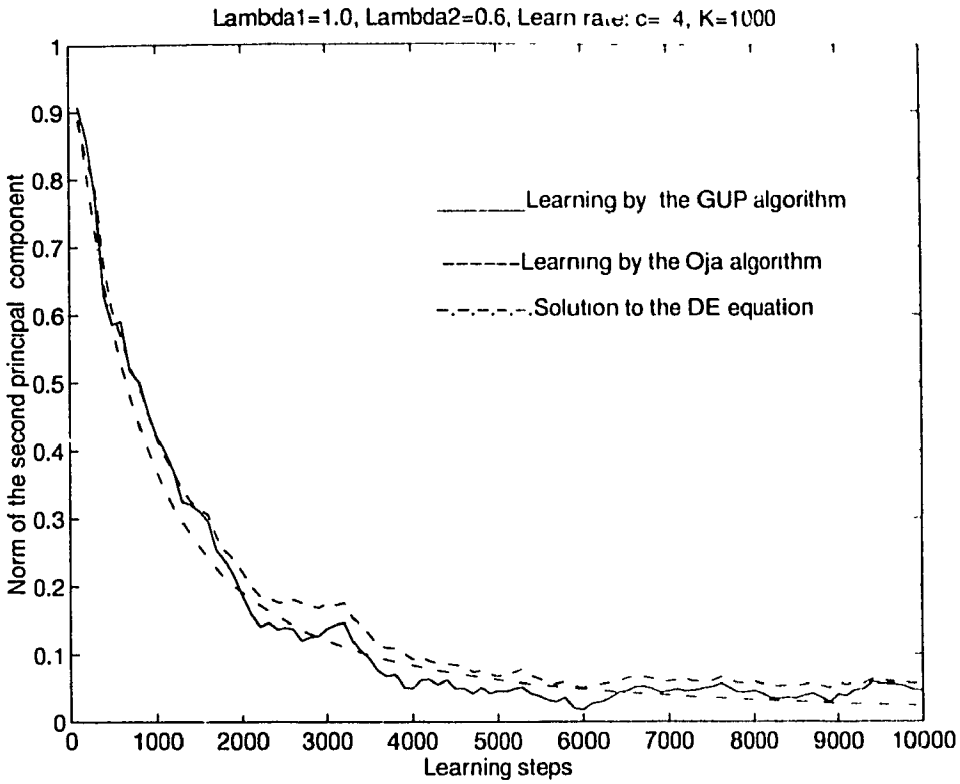

Figure 14: The learning results and the robustness of the GUP and Oja algorithms

Also, $C_x = E(XX^T)$ and $C_o = E(OO^T)$. The correlation matrix of Y is obtained as

$$C_y = E(YY^T) = (1-p)E(XX^T) + pE(OO^T) = (1-p)C_x + pC_o.$$

Usually, the eigenvectors of C_x are no longer the the eigenvectors of C_y. However, if C_o has the same eigenvectors as C_x, so has C_y. The first eigenvector of C_x corresponding to the largest eigenvalue may not be the first eigenvector of C_y corresponding to the largest eigenvalue of C_y.

More specifically, if $C_o = \lambda_o I$, the order of the eigenvectors of C_y still remains the same as that of C_x.

Case I: $C_o = \lambda_o I$

Without loss of generality, let $C_x = diag(\lambda_1, \lambda_2, ..., \lambda_N)$, where $\lambda_1 < \lambda_2 < ... < \lambda_N$. The correlation matrix of Y is

$$C_y = (1-p)C_x + p\lambda_o I = diag((1-p)\lambda_1 + p\lambda_o, (1-p)\lambda_2 + p\lambda_o, ..., (1-p)\lambda_N + p\lambda_0).$$

If the Oja learning algorithms are used, the learning algorithm still converges to the first eigenvector of C_x. Since $(1-p)\lambda_1 + p\lambda_o - (1-p)\lambda_2 - p\lambda_o = (1-p)(\lambda_1 - \lambda_2)$, the convergence speed of the learning algorithms is $O(n^{-c(1-p)(\lambda_1-\lambda_2)})$. We see that the convergence speed does not change very much when p is small. On the other hand, let D_{ky}, D_{kx}, D_{ko} be the actual noise, the sample noise, and the outlier noise respectively for the Oja learning algorithms with outliers. We have

$$E(||D_{ky}||^2) = (1-p)E(||D_{kx}||^2) + pE(||D_{ko}||^2) \approx (1-p)tr(C_x)\lambda_1 + ptr(C_o)\lambda_o$$

$$= (1-p)tr(C_x)\lambda_1 + pN\lambda_o^2.$$

The robustness of the Oja learning algorithms under outliers is obtained as

$$\overline{R}_{Oja} = \frac{K}{c^2((1-p)tr(C_x)\lambda_1 + pN\lambda_o^2)}.$$

We can see that the robustness is greatly reduced for a large p or λ_o. For example, let $p = 0.1$ and $\lambda_o = 10\lambda_1$. Then, $\overline{R}_{Oja} \approx \frac{1}{11}R_{Oja}$, where R_{Oja} is the robustness of the Oja algorithms for outlier free learning.

For the unsupervised Perceptron algorithms, assume that X and O have a normal distribution. B_1 and B_2 are the first two eigenvectors of C_x. We have

$$d_1 = E(|B_1^T Y|) = (1-p)E(|B_1^T X|) + pE(|B_1^T O|) = (1-p)\rho\sqrt{\lambda_1} + p\rho\sqrt{\lambda_o},$$

and

$$d_2 = E(|B_2^T Y|) = (1-p)\rho\sqrt{\lambda_2} + p\rho\sqrt{\lambda_o}.$$

For the GHP algorithm with outliers, the learning speed is approximately

$$O(\epsilon^{c(\frac{d_2}{d_1}-1)}) = O(\epsilon^{c((\frac{(1-p)\sqrt{\lambda_1}+p\sqrt{\lambda_o}}{(1-p)\sqrt{\lambda_1}+p\sqrt{\lambda_o}})^2-1)}),$$

which shows that the learning speed of the GUP algorithm decreases mildly. For the GUP algorithm,

$$E(||D_{ky}||^2) = (1-p)E(||D_{kx}||^2) + pE(||D_{ko}||^2)$$

$$\approx (1-p)tr(C_x) + ptr(C_o) = (1-p)tr(C_x) + pN\lambda_o.$$

The robustness of the GUP algorithm in the presence of outliers is

$$\overline{R}_{GUP} = \frac{K}{c^2((1-p)tr(C_x) + pN\lambda_o)}.$$

The robustness of the GUP algorithm also decreases, but the decreasing rate is much smaller than that of the Oja learning algorithms.

If we set $c' = c\frac{\lambda_2-\lambda_1}{\lambda_1}\frac{d_1}{d_2-d_1}$ as the learning rate of the GUP algorithm with outliers, the learning speed is the same as that of the outlier free learning algorithm with a learning rate parameter c. Then, the robustness of the GUP learning algorithm is

$$R_{GUP} = \frac{K}{(c')^2((1-p)tr(C_x) + pN\lambda_o)}. \tag{117}$$

Using the same example for the Oja learning algorithm ($p = 0.1$ and $\lambda_o = 10\lambda_1$), we obtain, $\overline{R}_{GUP} \approx \frac{1}{3}R_{GUP}$, where R_{GUP} is the robustness for the outlier free algorithm. Similar results can be derived for the NUP algorithm. Therefore, in the presence of outliers with correlation matrix $C_o = \lambda_o I$, the unsupervised Perceptrons are much more robust than the Oja learning algorithms. This is also demonstrated by the simulation results. Fig. 15 shows the distribution of the training data with outliers. Fig. 16 shows the learning results using the Oja and the GUP algorithms.

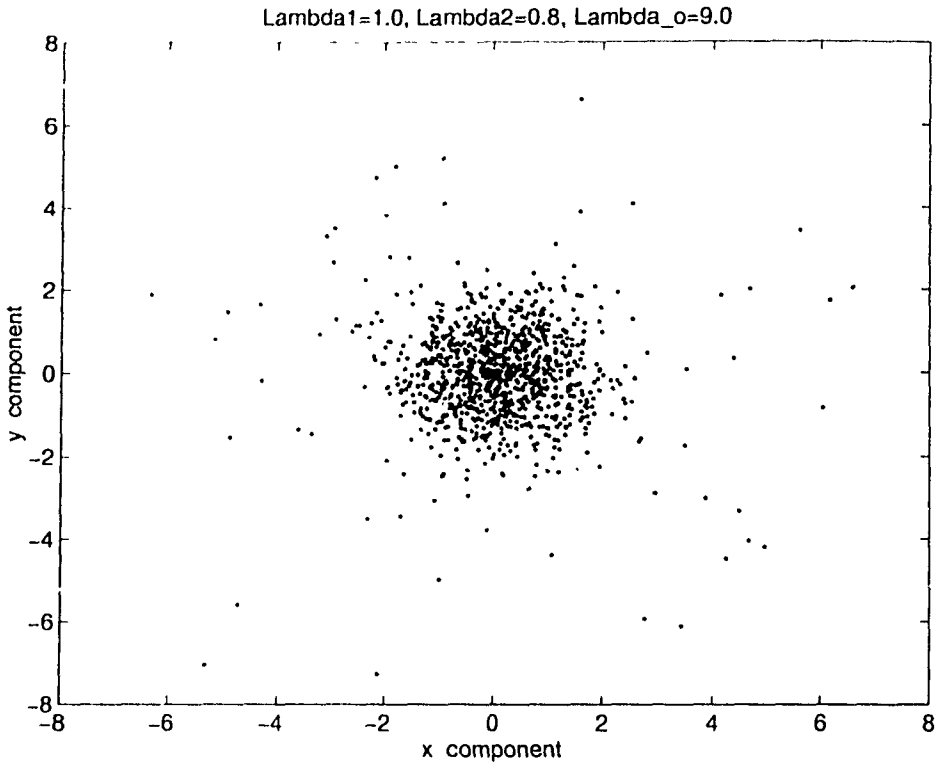

Figure 15: The distribution of the training data in the presence of outliers

Case II: $B^T C_o B = diag(\lambda_{o1}, \lambda_{o2}, ..., \lambda_{oN})$

For the Oja learning algorithms, the correlation matrix of Y is

$$C_y = (1-p)C_x + pC_o = diag((1-p)\lambda_1 + p\lambda_{o1}, (1-p)\lambda_2 + p\lambda_{o2}, ..., (1-p)\lambda_N + p\lambda_{oM}).$$

Let i be the index that maximizes $(1-p)\lambda_i + p\lambda_{oi}$. For $i > 1$, if

$$(1-p)\lambda_1 + p\lambda_{o1} < (1-p)\lambda_i + p\lambda_{oi},$$

which can be rewritten as

$$\lambda_{oi} - \lambda_{o1} > \frac{1-p}{p}(\lambda_1 - \lambda_i). \tag{118}$$

B_1 is no longer the first eigenvector of C_y corresponding to the largest eigenvalue. Hence, the learning algorithm with outliers no longer converges to the first principal component for the outlier free data.

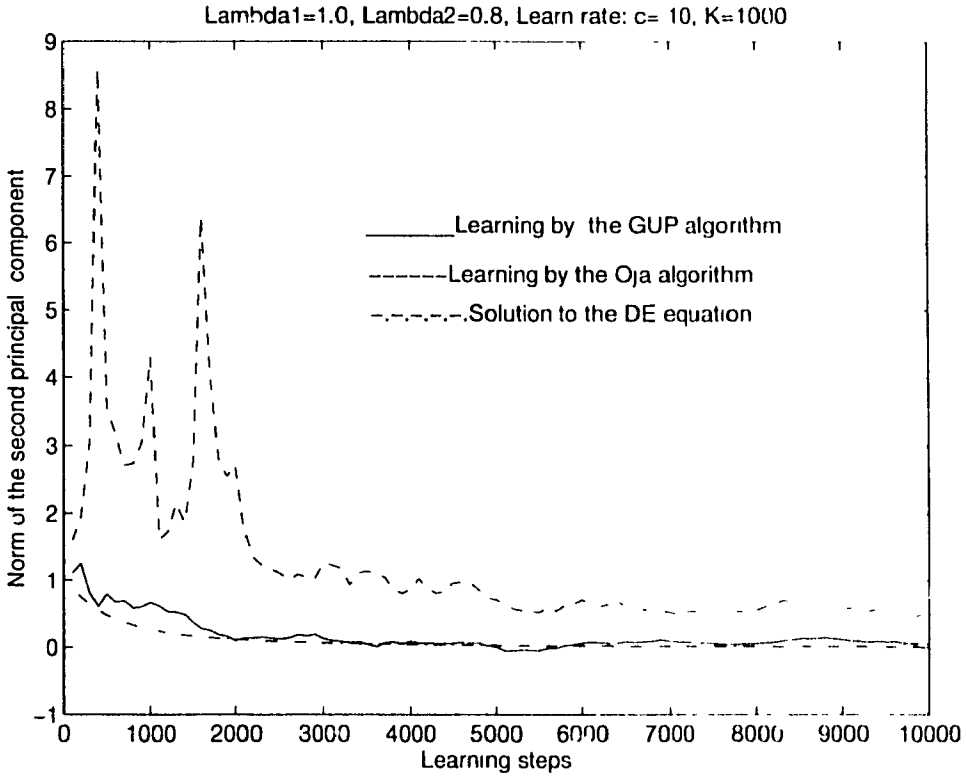

Figure 16: The learning results of the Oja and the GUP algorithms under the presence of outliers

For the unsupervised Perceptron algorithms, we have

$$d_1 = E(|B_1^T Y|) = (1-p)E(|B_1^T X|) + pE(|B_1^T O|) = (1-p)\rho\sqrt{\lambda_1} + p\rho\sqrt{\lambda_{o1}}$$

and

$$d_i = E(|B_i Y|) = (1-p)\rho\sqrt{\lambda_i} + p\rho\sqrt{\lambda_{oi}}.$$

The criterion $E(|W^T X|)$ with constraint $||W|| = 1$ has global optima at B_i, if and only if $d_i > d_1$. That is

$$(1-p)\rho\sqrt{\lambda_i} + p\rho\sqrt{\lambda_{oi}} > (1-p)\rho\sqrt{\lambda_1} + p\rho\sqrt{\lambda_{o1}},$$

or rewritten as

$$\lambda_{oi} - \lambda_{o1} > \frac{1-p}{p}(\lambda_1 - \lambda_i)\frac{\sqrt{\lambda_{oi}} + \sqrt{\lambda_{o1}}}{\sqrt{\lambda_i} + \sqrt{\lambda_1}}. \tag{119}$$

Usually, $\frac{\sqrt{\lambda_{oi}}+\sqrt{\lambda_{o1}}}{\sqrt{\lambda_i}+\sqrt{\lambda_1}} >> 1$. Thus, the condition (118) is much easier to be satisfied than the condition (119). In this sense, the unsupervised Perceptron algorithms are also more robust than the Oja learning algorithms.

If the condition (118) for the Oja learning algorithm and the condition (119) for the unsupervised Perceptron algorithm are not satisfied, the learning algorithms still converge to the first principal component. Similar to the Case I, it is easy to give the learning speeds and the robustness of the learning algorithms. Also, the unsupervised Perceptron algorithms are more robust than the Oja learning algorithms.

3.8 Outlier Analysis of supervised Learning Algorithms

For the supervised learning algorithms, both input data and outputs may contain outliers.

For an outlier input vector O and response y_o, let $C_o = E(OO^T)$ and $R_o = E(y_o O)$.

If $R_o = C_o \overline{Z}$, the LMS learning algorithm still converges to the optimal solution $\overline{Z}$. When the training data contain outliers, the convergence speed will increase and the robustness of the LMS algorithm is reduced. However, the overall performance of the learning algorithm does not change very much.

If $R_o \neq C_o \overline{Z}$, the LMS learning algorithm cannot converge to $\overline{Z}$. It converges to $(pC_x + (1-p)C_o)^{-1}(pR_x + (1-p)R_o)$, which can be quite different from $\overline{Z}$.

For example, for the two-class classification problem, the output value can be set as -1 or $+1$. Fig. 17 shows the optimal solution for the outlier free data. Under the presence of outliers, the performance of the LMS algorithm may be very poor.

If the following absolute criterion based increment correction algorithm (ICA),

$$W_{k+1} = \begin{cases} W_k + \gamma_k X_k & \text{if } W_k^T X_k - y_k < 0, \\ W_k - \gamma_k X_k & \text{otherwise} \end{cases} \tag{120}$$

is used, the performance of the classifier is still acceptable.

Fig. 18 shows the classifiers based on the LMS and increment correction algorithms. The increment correction algorithm is less sensitive to the outliers than the LMS algorithm.

3.9 Summary

Based on the analyses and the definitions given in Chapter 2, the convergence, convergence speeds, robustness, bias and variance of several single neuron based learning algorithms have been analyzed in this chapter.

Theoretically, the learning speed of a learning algorithm can be designed as fast as possible by choosing large enough learning rate parameters or scaling the training

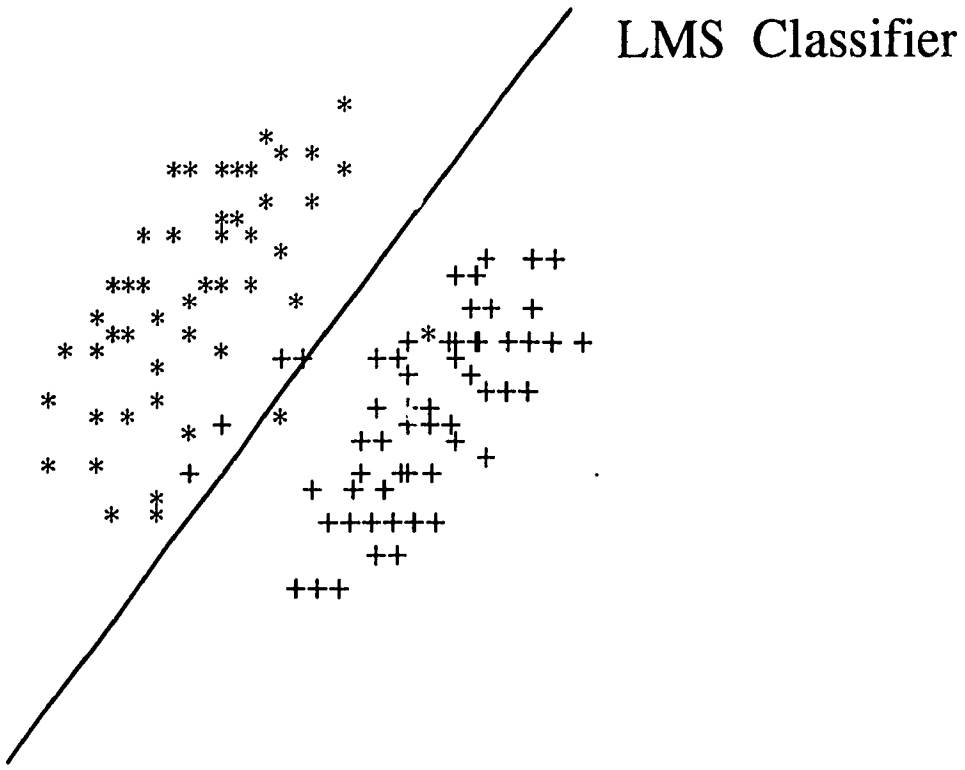

Figure 17: The LMS classifier for normal training data

data. However, the robustness of the learning algorithm will decrease with the corresponding rate. Hence, the convergence speed analysis and the robustness analysis must be used together.

A single neuron based learning algorithms can be derived or associated with a quadratic criterion or an absolute criterion. With the same input data, it is shown that the performances of the two classes of algorithms have no obvious differences in the outlier free environment. However, when the training samples contain outliers, the absolute criterion based learning algorithms are much more robust than the quadratic criterion based learning algorithm .

Based on the theoretical analysis of the bias and the variance of single neuron based learning algorithms, the optimal learning rate parameters can be found. However, since the eigenvalues of input data are usually unknown before learning, the

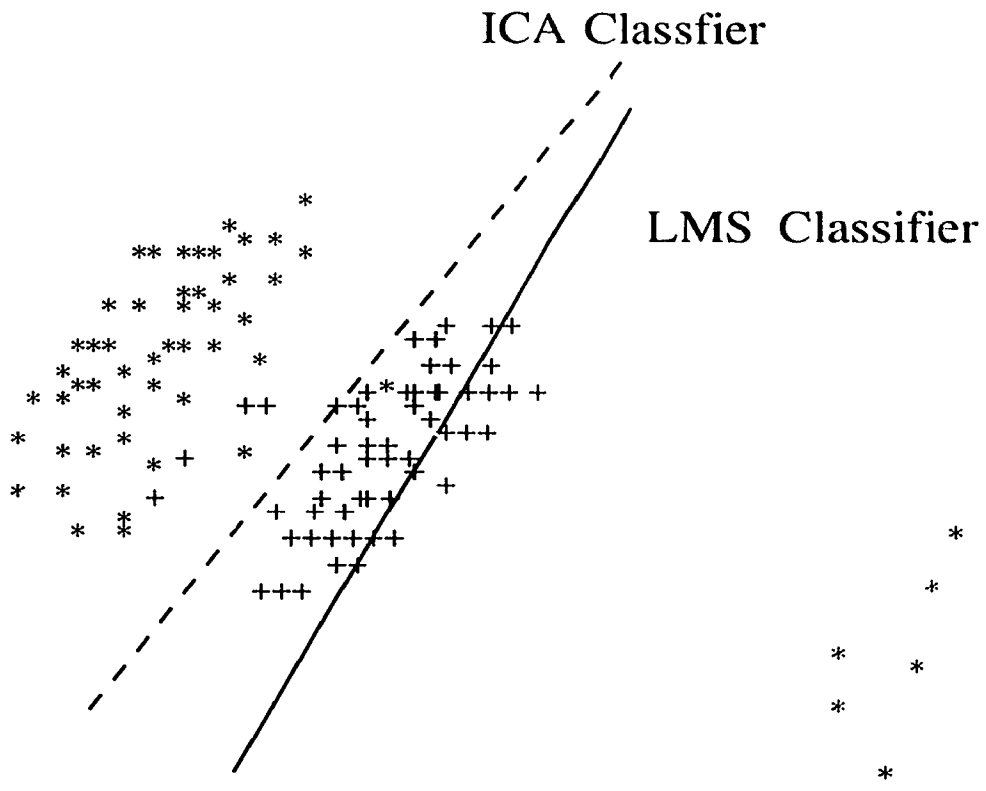

Figure 18: The LMS classifier and absolute criterion based classifier in the presence of outliers

optimal learning rate parameters are difficult to obtain for a new set of training samples. Using the GUP algorithm as an example, we show that there is a good agreement between the theoretical results and the simulation results for choosing an optimal learning rate.

Chapter 4

Optimal Binary Tree Classifiers

4.1 Introduction

Tree classifiers are widely used in pattern recognition and machine learning. Such classifiers have the advantage of fast pattern recognition speed while preserving the same representative capability as the multi-layer Perceptrons. The tree classifier may also provide insight into the structure of data. Because of these advantages, tree classifiers have received a lot of attention in the literature [1, 9, 10, 11, 38, 15, 60]. A recent survey is found in [68].

Tree classifiers can be generated through either supervised or unsupervised learning. Most tree classifiers are generated using supervised learning techniques. In this chapter, we consider both approaches. In particular, we are concerned with binary classification trees only.

Unlike CART and traditional decision trees [7, 10, 22], our tree classifier uses the entire set of attributes in sample patterns. We adopt the unsupervised Perceptron to train an unsupervised tree classifier and the Fisher's linear discriminant to train a supervised tree classifier. Each node of the tree is divided by a hyperplane and overlapping of patterns is allowed to within a fixed error measure so as to improve classification accuracy.

4.2 The Construction of the Binary Tree Classifier

A tree classifier attempts to match a given input pattern against a set of model patterns in order to find the model pattern which is most similar to the input pattern. Some similarity measure is used in pattern matching. Given an input pattern X and a tree classifier with a set of leaf nodes $\{L_1, L_2, ..., L_n\}$, one desires to find the leaf node L_i such that the error $\|X - L_i\|$ is the minimum for $1 \leq i \leq n$. Since a path to any leaf node in the tree may contain intermediate nodes, classification errors may occur during the search of the leaf nodes if only limited backtracking is allowed. We aim at minimizing such classification errors.

There are two basic approaches to constructing tree classifiers, namely, the agglomerative and the divisive approaches [71]. We employ the divisive approach in our algorithm. Specifically, our method works in the following manner: for the set of all sample patterns which are selected from a given distribution, a root node is constructed to cover all sample patterns. The root node is then divided into two subnodes, each of which represents a subset of samples. This process continues by further dividing the leaf nodes until certain criteria are met. To partition a set of sample patterns, we use a hyperplane to cut the sample space into two parts with overlap as shown in Fig.19.

4.2.1 The Tree Construction Algorithm

Let Q be a FIFO (first in first out) queue, the divisive algorithm for constructing a binary tree classifier is formally described below.

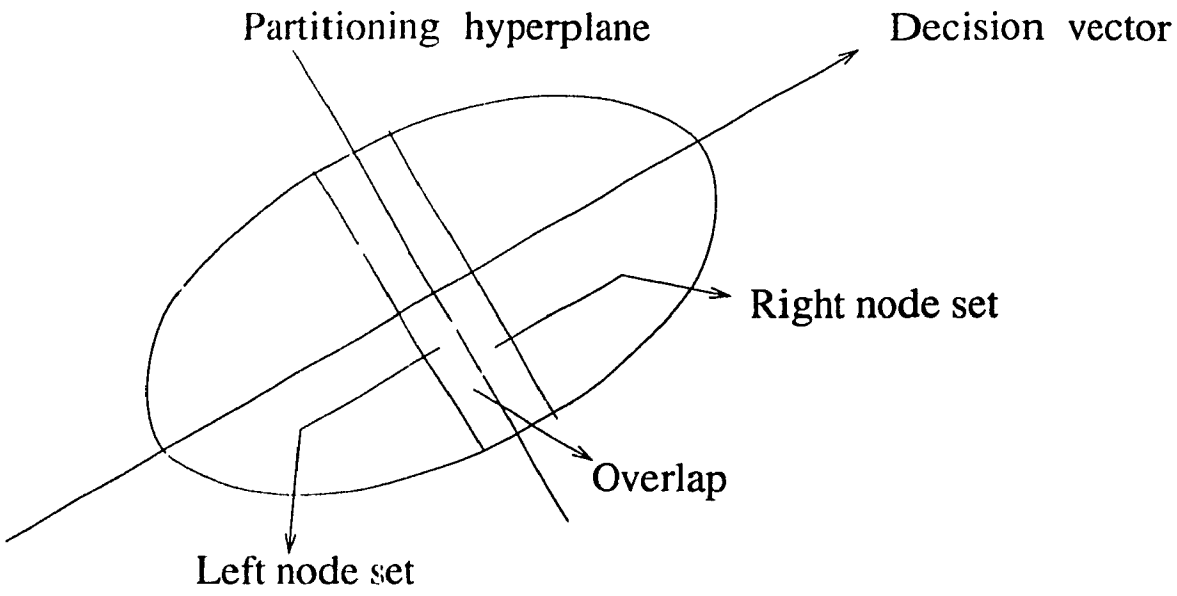

Figure 19: The partitioning hyperplane at a node of tree classifier

Binary Tree Classifier Algorithm

Step 1 : Create a root node and assign all the sample patterns to the root node. Place the root node on Q.

Step 2 : If Q is empty, go to Step 8. Otherwise, dequeue a node from Q.

Step 3 : If the number of the sample patterns covered by the node is less than M, go to Step 2.

Step 4 : Find the maximum eigenvector of the covariance matrix of the sample patterns covered by the current node and find the mean vector of the sample patterns.

Step 5 : Construct a hyperplane whose norm is the eigenvector and goes through the mean point of the sample patterns.

Step 6 : Using the hyperplane constructed in Step 4, separate the sample patterns

into two subsets and create two nodes for each subset with overlap.

Step 7 : Place both new nodes on Q and go to Step 2.

Step 8 : End.

In step 4 above, the eigenvector of the maximum eigenvalue corresponds to one principal axis which has the largest variation. This is similar to the principal component analysis in numerical taxonomy [71].

4.2.2 Learning the Principal Component and the Mean

Our unsupervised learning algorithm needs the principal eigenvector for a set of sample patterns. We employ the unsupervised Perceptron algorithm for learning the principal eigenvector and the mean. If W is a vector that is perpendicular to the partition hyperplane H, W is chosen as the eigenvector of the maximum eigenvalue for the covariance matrix of samples. The eigenvector can be computed from the covariance matrix of samples. On the other hand, the eigenvector can be learned from samples using neural network methods. Several methods have been proposed for learning the eigenvector from sample patterns [56, 69]. We present here a slight modification of the unsupervised Perceptron algorithm.

Assume that M is the center of the samples. M can be trained using the incremental rule given below. Let M_1 be an initial guess of the mean value and M_i be the approximation at step i. Given a new sample pattern X_i, the next approximation M_{i+1} is given by

$$M_{i+1} = \frac{iM_i}{i+1} + \frac{X_i}{i+1}.$$

This can be shown to converge to the mean value [78]. The eigenvector can be trained by the following unsupervised Perceptron algorithm:

$$W_{t+1} = \begin{cases} W_t + (X_t - M_t) & \text{if } W_t^T(X_t - M_t) > 0, \\ W_t - (X_t - M_t) & \text{otherwise.} \end{cases} \quad (121)$$

It is also known that the principal component is the first coordinate of the Karhunen-Loeve (K-L) expansion. In the past, FFT, Harr and Walsh transforms were often used as the feature selection and extraction for constructing decision trees. The K-L transform was not widely used in the recognition of large character sets. The main reasons against the use of K-L transform are given below.

1. It would have to be computed at each internal node of the tree in the design phase. Furthermore, much computation would be necessary for unknown pattern to pass through each level of the tree in the recognition phase.
2. The information about each concrete K-L transform must be stored in the internal node, which would take up much more memory than other structures built with other transforms.

In our approach, only the principal eigenvector is computed and we have an efficient learning algorithm for the principal eigenvector. Therefore the computational inefficiency of the K-L transform is overcome in our approach. Furthermore, our approach requires only a vector inner product which can be done quickly and efficiently. If limited backtracking is used in the search of matching patterns, there is no need to store the transformed patterns. Hence the storage inefficiency problem is also absent in our approach.

4.3 Analysis of the Classifier

In this section, we first analyze the classification error of our tree classifier. Then we will study the relationship between the tree structure and various parameters.

4.3.1 Classification Error Analysis

Let M and C be the mean vector and the covariance matrix of samples respectively. Let $P(X, W)$ represent the projection of X along the direction W. $P(X, W)$ is also a random variable. If X has a multi-variate normal distribution, $P(X, W)$ is a normal variable. Furthermore, $P(X, W)$ has a symmetric distribution with respect to its mean $E[P(X, W)]$ if X has a symmetric distribution with respect to M. Its variance is given by

$$\sigma_W^2 = \frac{W^T C W}{||W||^2}. \tag{122}$$

With W as the norm, we can construct a hyperplane H such that H passes through the mean point M. We use this hyperplane to divide the sample set into two subsets C_1 and C_2, possibly with an overlap. So W is the norm of the hyperplane and M is the mean point of the sample patterns. Hence we have the hyperplane

$$W^T(X - M) = 0. \tag{123}$$

Since $P(X, W)$ is the projection of $X - M$ along direction W, we arrive at

$$P(X, W) = \frac{W^T(X - M)}{||W||}. \tag{124}$$

If we allow the two subsets to have some degree of overlap, we have, for a given overlap length $\alpha > 0$, the partition rules

If $P(X, W) \geq \alpha, X \in C_1$

If $P(X, W) \leq -\alpha, X \in C_2$

If $-\alpha < P(X, W) < \alpha, X \in C_1 \bigcap C_2$.

The set $C_1 \cap C_2$ is the overlap of the two subsets. For normally distributed X, the probability of the overlap is

$$P_o(\alpha) = \frac{1}{\sqrt{2\pi}\sigma_W} \int_{-\alpha}^{\alpha} e^{-\frac{x^2}{\sigma_v^2}} dx = 2[1 - \Phi(\frac{\alpha}{\sigma_v})], \tag{125}$$

where $\Phi(x)$, the probability that y is less than x is given by

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-y^2} dy.$$

Therefore, if α is a constant, $P_o(\alpha)$ is a monotonic increasing function of σ_v. When σ_v is maximum, $P_o(\alpha)$ is minimum.

According to (122), when $W = B_1$, $max(\sigma_W) = \lambda_1$, where λ_1 is the largest eigenvalue of covariance matrix C and B_1 is the eigenvector corresponding to eigenvalue λ_{max}. We assume that X is a normally distributed multivariate random vector with expectation M and covariance Σ. W is the normalized eigenvector. The hyperplane $W^T(X - M) = 0$ splits the space into two parts with an overlap α. For any constant α, when W is the eigenvector corresponding to the maximum eigenvalue λ_1. The probability of overlap is minimum, P_{min}.

$$P_{min} = 2[1 - \Phi(\frac{\alpha}{\lambda_{max}})]. \tag{126}$$

When a noise signal is added to an input pattern, a node may misclassify the pattern. Assume that the query pattern X is produced by adding noise to one of the model patterns Y and that X is a random multivariate distributed normally with mean vector Y and covariance $\sigma_n I$, where I is the identity matrix. The probability that X and Y are divided into different sides of the hyperplane is

$$P(y) = \frac{1}{\sqrt{2\pi}\sigma_n} \int_{\alpha+y}^{\infty} e^{-\frac{x^2}{\sigma_n^2}} dx. \tag{127}$$

Because $y = P(Y,W)$ is a normally distributed variate, the partition error for all sample patterns is

$$P_e = 2\frac{1}{\sqrt{2\pi}\sigma_W}\int_0^\infty P(y)e^{-\frac{y^2}{\sigma_W^2}}dy = \frac{1}{\pi\sigma_n\sigma_W}\int_0^\infty\int_{\alpha+y}^\infty e^{-\frac{x^2}{\sigma_n^2}-\frac{y^2}{\sigma_W^2}}dxdy. \qquad (128)$$

Therefore, the partition error is a function of σ_W, σ_n and α. If α is a constant, in the direction that makes σ_W maximum, the probability of partition error is minimum. That is, when the largest eigenvector is selected as partition direction, the partition error is minimized.

4.3.2 Tree Structure Analysis

Because the classification error accumulates level by level in the decision tree, it is important to employ backtracking in the search of matching patterns. However, backtracking implies increased computational complexity. It is thus desirable to minimize the backtracking effort. By allowing limited overlapping of patterns, backtracking can be efficiently managed. In the analysis below, we assume limited overlapping is employed.

Let the total number of the samples be M and assume that the number of patterns in each terminal node is equal to or less than T. L_e represents the expected number of layers of the tree and p_o is the frequency of overlapping. We have the following relationship:

$$M(\frac{1+p_0}{2})^{L_e} \approx T.$$

The expected number of layers is thus

$$L_e \approx \frac{log(\frac{T}{M})}{log(\frac{1+p_o}{2})} = \frac{logM - logT}{log2 - log(1+p_o)}. \qquad (129)$$

The expected number of nodes in the tree is approximately

$$N_r \approx 2 * 2^{L_r} - 1 \approx 2 * 2^{\frac{\log M - \log I}{\log 2 - \log(1+p_o)}} - 1. \quad (130)$$

The expectation of the number of the nodes is a function of the number of terminal nodes, the length of overlap, and the number of model patterns. For each node, the probability of overlap is minimum for a constant overlap length using the above split rule. Therefore the nodes of the tree are expected to be minima. Increasing the overlap length can quickly increase the number of nodes in the tree and largely improve match accuracy. However, a penalty is paid for having to search more nodes and slowing down the computation.

4.4 Supervised Tree Classifier

The tree classifier given above can also be extended to the supervised case. If there are only two classes, the partition hyperplane at each node is easily obtained by traditional statistic classification algorithms, the Perceptron, or the Windrow-Hoff learning algorithm. For patterns of multiple classes, these algorithms cannot be directly applied to find the optimal hyperplane. A hyperplane is perpendicular to a one-dimensional line. From the feature selection point of view, to find a hyperplane is to reduce the dimension of sample vectors to one or select the best feature for samples. Then, we split the samples based on the selected line or feature. Fisher's multiple linear discriminant can be applied to construct a binary tree classifier for multiple class samples.

4.4.1 Tree Classifier Based on Fisher's Multiple Linear Discriminant

Fisher's linear discriminant can reduce the dimension n to m ($m < n$)[17] for multiple class training samples. In order to construct a tree classifier, we merely project the n-dimensional data onto a line. Thus, we choose m to be 1.

Assume that there are N training samples, $X_1, X_2, ...X_N$. The samples belong to c classes, $\omega_1, \omega_2, ..., \omega_c$, n_i is the number of samples in class ω_i.

The within-class scatter matrix is given by

$$S_W = \sum_{i=1}^{c} S_i,$$

where,

$$S_i = \sum_{X \in \omega_i} (X - m_i)(X - m_i)^T,$$

and

$$m_i = \frac{1}{n_i} \sum_{X \in \omega_i} X.$$

The between-class scatter matrix is defined as

$$S_B = \sum_{i=1}^{c} n_i (m_i - m)(m_i - m)^T,$$

and

$$m = \frac{1}{N} \sum_{i=1}^{c} \sum_{X \in \omega_i} X = \frac{1}{N} \sum_{i=1}^{c} n_i m_i.$$

The projection from an n-dimensional space onto a one-dimensional space is accomplished by a discriminant function

$$\tilde{X} = W^T X,$$

where W is an n-dimensional vector. The samples $X_1, X_2, ..., X_N$ project to a corresponding set of samples $\tilde{X}_1, \tilde{X}_2, ..., \tilde{X}_N$ which can be described by their own mean vectors and scatter matrices. Thus, if we define

$$\tilde{m}_i = \frac{1}{n_i} \sum_{\tilde{X} \in \omega_i} \tilde{X},$$

$$\tilde{m} = \frac{1}{N} \sum_{i=1}^{c} n_i \tilde{m}_i,$$

$$\tilde{S}_W = \sum_{i=1}^{c} \sum_{\tilde{X} \in \omega_i} (\tilde{X} - \tilde{m}_i)(\tilde{X} - \tilde{m}_i)^T,$$

and

$$\tilde{S}_B = \sum_{i=1}^{c} n_i (\tilde{m}_i - \tilde{m})(\tilde{m}_i - \tilde{m})^T.$$

It is straightforward to show that

$$\tilde{S}_W = W^T S_W W,$$

and

$$\tilde{S}_B = W^T S_B W.$$

These equations show how the within and between-class scatter matrices are transformed by the projection onto the one-dimensional space. What we seek is a transformation vector W that in some sense maximizes the ratio of the between-class scatter to within-class scatter. Using this measure, we obtain the criterion function,

$$J(W) = \frac{\tilde{S}_B}{\tilde{S}_W} = \frac{W^T S_B W}{W^T S_W W}.$$

If $|S_W| \neq 0$, W is the generalized eigenvector that corresponds to the largest eigenvalue in

$$S_B W = \lambda S_W W.$$

The eigenvalue λ_1 is the largest root of the characteristic polynomial

$$|S_B - \lambda S_W| = 0.$$

We can solve

$$(S_B - \lambda_1 S_W)W = 0$$

for the eigenvector of W.

Therefore, the n-dimensional sample data are projected onto a line. To construct a binary tree classifier, we split the samples at the center of the line as following:

If $W^T X - \tilde{m} > 0, X$ is assigned to the left node. Otherwise, X is assigned to the right node.

4.5 The Search Algorithm for Classification

The search algorithm is dependent upon the overlapping length. When the overlapping length is 0, the search algorithm explores only a single path to a leaf node, and the search time is the shortest. Otherwise, the search algorithm may have to explore several leaf nodes through a number of search paths. When the overlapping length increases, the search time also increases. On the other hand, the classification error decreases as the overlap increases. Backtracking is used to explore several paths. Let α be the overlapping length for constructing the tree classifier and let β be the vigilance factor for searching the tree. When the error is less than the vigilance factor, the search can be stopped. In addition, we define a search threshold γ. During the search, we accumulate the error on each level. When the accumulative error is less than γ, that path must be explored. In the search algorithm, we use a queue Q to

store nodes. Let $S_0 = 0$ be the initial accumulative error and S_i be the accumulative error of a node on the i-th level. Given an input pattern X, the generic search algorithm is described below.

Step 1 $W \leftarrow root$;

$e_W \leftarrow +||V_W^T(X - M_W)||$

enqueue(W, Q).

Step 2 If Q is empty, exit and no matching pattern is found;

Otherwise, $W \leftarrow$ dequeue(Q).

Step 3 If W is a leaf node and $||X - P_W|| < \beta$, exit and classify X as in the same class as W.

Step 4 If W is not a leaf node,

locate its right child R and left child L;

$e_R \leftarrow e_W + ||V_R^T(X - M_R)||$;

$e_L \leftarrow e_W + ||V_L^T(X - M_L)||$.

Step 5 If $V_R^T(X - M_R) > 0$, then enqueue(R, Q);

If $e_L < \gamma$, then enqueue(L, Q).

Step 6 If $V_R^T(X - M_R) \leq 0$, then enqueue(L, Q);

If $e_R < \gamma$ then enqueue(R, Q).

Step 7 Go to Step 2.

In the above search algorithm, M_W and V_W are the mean of the patterns in node W and the principal eigenvector of the covariance matrix of node W, respectively. P_W is a model pattern stored in a leaf node W.

If $\gamma = 0$, the search algorithm explores a single path to a root node. Let L_e be the expected number of levels of the tree. The expected number of nodes to be explored is simply L_e. Hence the search time C_t is proportional to $O((L_e + T)N)$, where N is the dimension of the pattern vector.

$$C_t = O((L_e + T) * N) = O(N \frac{log(\frac{T}{M})}{log(\frac{1+P_o}{2})}) = O(N \frac{log M - log T}{log 2 - log(1 + p_o)} + NT). \quad (131)$$

When T decreases, C_t decreases and the nodes of the tree increases. If the overlap length is decreased, the number of nodes of the tree is decreased and the search time is decreased. However, the classification error will increase.

4.6 Results of Experimental Study

We present results of experimental study in this section. We have applied our approach to waveform recognition and Chinese character recognition. The results are summarized in the following subsections.

4.6.1 Results for Waveform Recognition

CART (Classification and Regression Trees) has been extensively studied in the literature [7, 68]. The waveform recognition problem was used to compare various optimization schemes for CART. Therefore we also use this problem as a benchmark to compare with other approaches.

The waveform recognition problem is based on three waveform functions $h_1(t)$, $h_2(t)$, $h_3(t)$ as shown in Fig.20. The measurement vectors are 21-dimensional, that is, $\mathbf{X} = (x_1, ..., x_{21})$. To generate a class 1 vector $\mathbf{X}$, we independently generate a uniform random number μ and 21 other random numbers $e_1, e_2, ..., e_{21}$. e_i, where $1 \leq i \leq 21$, has a normal distribution with zero mean and unit variance. The components of a vector are computed according to

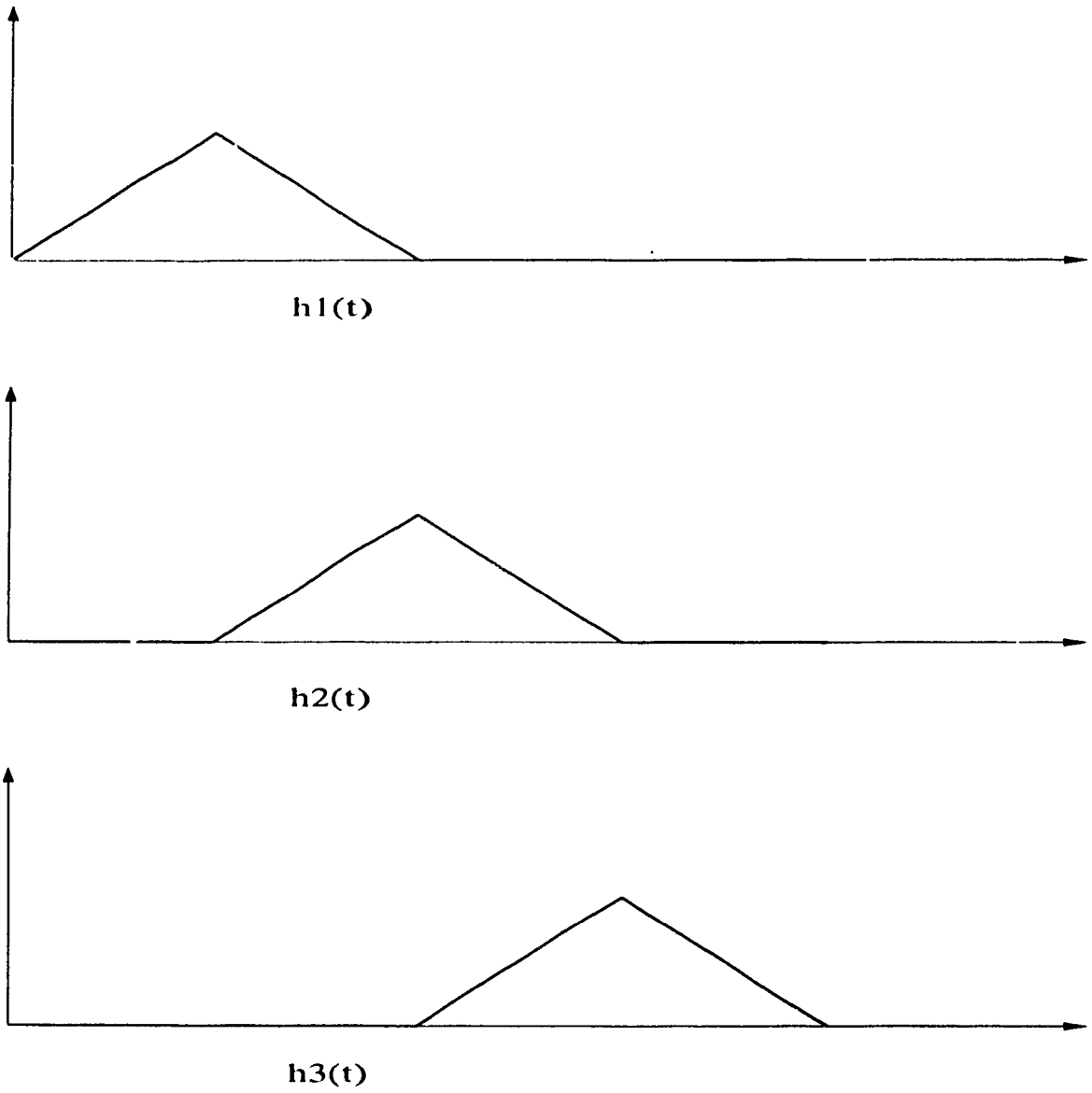

Figure 20: The waveform functions

$$x_i = \mu h_1(i) + (1-\mu)h_2(i) + e_i, i = 1, 2, ..., 21.$$

To generate a class 2 vector, repeat the preceding procedure and set

$$x_i = \mu h_1(i) + (1-\mu)h_3(i) + e_i, i = 1, 2, ..., 21.$$

A class 3 vector is generated by

$$x_i = \mu h_2(i) + (1-\mu)h_3(i) + e_i, i = 1, 2, ..., 21.$$

The CART tree was constructed using the original CART algorithm. The resubstitution estimate of misclassification is 0.14 and the estimate of misclassification by using an independent test set of size 5000 is 0.28. Using the linear variable combination method, a three node tree was constructed using CART. The error rate of the tree is 0.20.

In our experiments, 5000 samples are used to construct the tree classifier. The tree has 9 nodes. The misclassification rate is 0.15 for the samples that are used in constructing the tree. The misclassification rate is 0.16 using an independent test set of size 10000. Our experimental result is much better than that of CART.

4.6.2 Results for Chinese Character Recognition

Tree classifiers were constructed for recognizing three thousand printed Chinese characters. Each character is represented by a 50*50 pixel image. If the sum of each 5*10 block of pixels is considered as a feature of the character, a total of 50 features are selected for each character. One model pattern is selected for each character. Fig. 21 shows the projections of the 3000 characters at the first principal component. Fig. 22 illustrates the distribution of the projection.

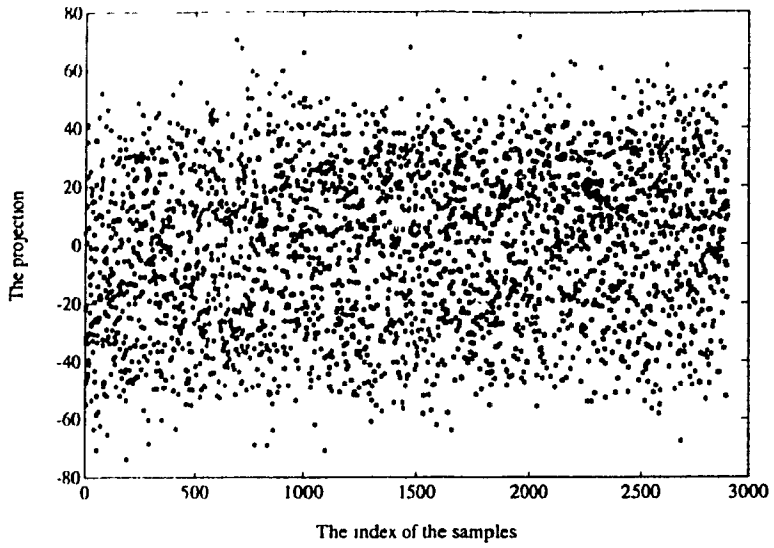

Figure 21: The projections of 3000 Chinese characters at the first principal component

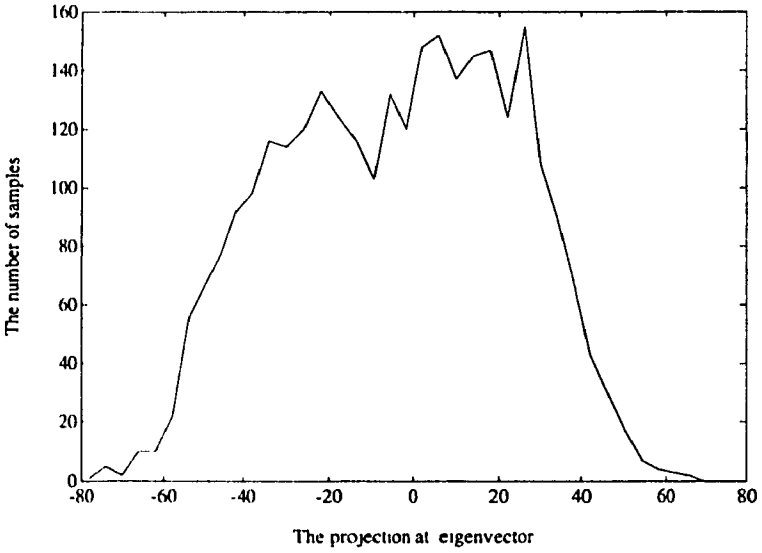

Figure 22: The distribution of the projections of character samples

If α is 5.0 and each terminal node contains $T < 20$ model patterns, the classification tree has 2173 nodes. Experimental results are summarized in the following tables.

$\alpha = 4.0, T_e = 20, S_n = 3000$

Noise	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0
Vig=0	100.0	99.8	98.1	93.3	84.5	77.5	69.0	62.1	55.0	47.9
Vig=50	100.0	99.8	99.5	98.7	98.9	98.4	95.9	93.0	86.2	79.0

Table 3: Recognition rates for different noise and search vigilance

In these tables, α is the overlapping length, T_e is the maximum number of the samples allowed in a leaf node, and S_n is the total number of the training samples.

$\sigma_n = 3.3, T_e = 20, S_n = 3000$

α	0.0	1.0	2.0	3.0	4.0	5.0	6.0
No. of Nodes	447	525	779	1007	1457	2173	3461
Recognition Rate	72.20	80.70	87.1	93.50	95.54	98.5	99.67

Table 4: The number of nodes in the tree and the recognition rates for the different overlaps

$\alpha = 4.0, \sigma_n = 3.3, S_n = 3000$

T	20	30	40	50	60	70	80	90	100	500
L_e	1457	873	567	457	375	273	239	227	199	27
Reco. Rate	95.5	95.7	96.2	96.2	96.4	96.7	96.9	96.96	97.1	98.6

Table 5: The number of nodes in the tree and the recognition rate for the different numbers of maximum terminal node patterns

4.7 Summary

In this chapter, we present a new approach for constructing decision classifiers. Our approach combines the incremental learning of neural networks and the advantages

$\alpha = 7, \sigma_n = 5, T_e = 30$

S_n	500	1000	1500	2000	2500	3000
L_t	147	429	841	1433	2119	2991
Reco. Rate	96.8	94.4	94.2	92.9	92.88	92.83

Table 6: The number of nodes in the tree and the recognition rate for different numbers of training samples

$\alpha = 0, \sigma_n = 5, T_e = 3000$

σ_n	4.0	5.0	6.0	7.0	8.0	9.0	10.0
Reco. Rate	100	99.93	99.56	99.03	97.50	94.23	89.27
$\alpha = 7.0 vig = 50$	99.96	99.6	99.30	97.50	94.46	89.30	82.00

Table 7: The recognition rate for searching all samples and the tree classifier

of principal component analysis. The decision trees constructed using our approach show significant performance improvements.

We take advantage of the principal component analysis. However, our approach is different from principal component analysis which simply transforms the data from one space to another orthogonal space. We first find the principal axis and then use it to divide the sample patterns into two groups. This continues until certain termination criteria are satisfied.

On the other hand, each node in our tree classifier corresponds to an artificial neuron. The unsupervised Perceptron algorithms are employed to train the neurons in our trees. This is different from the traditional decision tree construction algorithms.

In addition, using the Fisher's linear discriminant, our approach is extended to construct a supervised tree classifier for multi-class training samples.

We have conducted many experiments to study the performance of our approach. Simulation experiments demonstrate the superb performance of our classification trees. Our approach presents an alternative technique to tree structured classifiers.

For Chinese character recognition, we mainly focus on the construction of a good classifier and an efficient search algorithm in this thesis. There is also a lot of work to be done to develop a real OCR system in the future.

Chapter 5

Asymmetric Associative Memory Model

5.1 Introduction

In recent years, artificial neural network have been shown to be a promising model of associative memory. From a distorted image or incomplete image, the network can associate a clear and complete one that has been stored. The Hopfield network has the capability of simulating associative memory [31, 32]. The Hopfield model is a symmetric network. All nodes in the network are connected with each other. The weight from node i to node j is equal to the weight from node j to i in the Hopfield network, that is, $w_{ij} = w_{ji}$. The weights of the network are given by the Hebbian rule which is realized by the output product method. The behavior of the network is described as a high-dimensional nonlinear dynamic system.

From an arbitrary initial state, when the elapsed time is long enough, the state that a system can attain is called an attractor of the dynamic system. If an attractor is a sample, we call it a sample attractor, otherwise, a non-sample attractor. Assume the initial state of a nonlinear dynamic system is set to a pattern X. The system finally attains an attractor. If the attractor is one of the samples that has been memorized,

it is considered that the sample is associated by X. The Hopfield network has two major limitations as an associative memory model. First, the samples that can be stored and accurately recalled are very limited. If a large number of samples are stored, many nonsample attractors will be produced, while some samples may not be the attractors of the network. Hopfield showed that the number of samples (M) that can be stored in a network is less than 0.15 times the number of nodes (N) in the network for randomly generated samples.

Another limitation is that the network cannot distinguish very similar samples. When the initial state of the network is set to a sample, the associated result may not be itself, but another similar sample.

In [59, 34], a nonlocal learning method was given to improve the memory capacity of the Hopfield model. However, the nonlocal learning rule makes the model unattractive from a biological viewpoint.

In this chapter, we propose an asymmetric associative memory neural network which is trained by the Perceptron algorithm. This associative memory network has a larger capacity when compared with the Hopfield network. In addition, a deepening impression method is given to enhance the performance of the associative memory.

5.2 Asymmetric Associative Memory Model

Although the Hopfield networks have some similar characteristics when compared with human associative memory, there are also some differences. For example, the Hopfield network learns only once for a sample while a human being often learns many times to memorize a sample. The connections between two nodes are symmetric in the Hopfield network. The symmetric connections cannot represent the internal structure of some information. Most real neural networks in a brain have

asymmetric connections. The relation between two components of a sample is often not symmetric.

In this section, an associative memory model is given whose performance is superior to the Hopfield network. The model is also a network with nodes connected with each other, but the connections are asymmetric. That is, there are two weights between node i and node j, W_{ij} is the weight which sends information from i to j . W_{ji} is the weight through which node i receives information from j. The two weights are usually different. The behavior of such a network is described by a nonlinear dynamic system. For a given pattern $X(0)$, considered as an initial state of the system, after k steps of operation, let the state of the system be $X(k)$. The evolution equation is

$$x_j(k+1) = f_h(\sum_{i \neq j} w_{ij} x_i(k)), \tag{132}$$

where $f_h(x)$ is the hard limiter nonlinearity

$$f_h(x) = \begin{cases} 1 & \text{if } x \geq 0, \\ -1 & \text{otherwise.} \end{cases}$$

Finally, the network moves to an attractor state. If the attractor is a sample attractor, we consider the sample to be associated by the pattern $X(0)$. We see that the evolution equation of this network is the same as that of the Hopfield network.

Such a network can be trained using the Ho-Kashyap [74] algorithm which is convergent if a solution exists and can yield the least mean square error solution. However, it is required to compute the pseudo-inverse of a matrix before learning. That is a very unnatural procedure when compared with human memory. Here we use the Perceptron learning algorithm to train the network. A component of a sample vector may be either $+1$ or -1. Assume that there are M sample vectors arranged as a periodic sequence with a period of M.

$$X(0), X(1), ..., X(t), ...$$

where vector $X(Mn+k)$ is the same as the kth sample vector. For sample $X(t)$, the weights are adapted using the following equation:

$$w_{ij}(t+1) = w_{ij}(t) + \eta[x_j(t) - f_h(\sum_{k \neq j} w_{kj}(t)x_k(t))]x_i(t) \tag{133}$$

where η is a constant. When no weight is changed after presentation of an entire period of M samples, the learning process is considered convergent and the learning ends. The following theorem shows that if a set of samples is well distributed in the pattern space and the number of samples is less than the number of the network nodes, the network can memorize all of them.

Theorem 5.1 *Given M binary samples, each sample is composed of N components. If the vectors composed of any $N-1$ components of the M samples are linearly independent in the corresponding $N-1$ dimensional Euclidean space, and $M < N$, then the above learning process is guaranteed to converge. All samples can be made stable attractors of the network.*

Proof: Assume the M samples are

$$X_i = (x_{i0}, x_{i1}, ..., x_{i,n-1}) \qquad (i = 1, 2, ..., M).$$

For any node of the network, for instance node 0, a group of learning equations are constructed as follows :

$$\begin{cases} w_{10}x_{11} + w_{20}x_{12} + ... + w_{N-1,0}x_{1,N-1} = x_{10}, \\ w_{10}x_{21} + w_{20}x_{22} + ... + w_{N-1,0}x_{2,N-1} = x_{20}, \\ \qquad\qquad ....., \\ w_{10}x_{M1} + w_{20}x_{M2} + ... + w_{N-1,0}x_{M,N-1} = x_{M0} \end{cases} \tag{134}$$

where w_{j0} is the weight from node j to node 0 which is considered as a variable. The determinant matrix of (134) is

$$\begin{vmatrix} x_{11} & x_{12} & \cdots & x_{1,N-1} \\ x_{21} & x_{22} & \cdots & x_{2,N-1} \\ \cdots & \cdots & \cdots & \cdots \\ x_{M1} & x_{M2} & \cdots & x_{M,N-1} \end{vmatrix}.$$

Because the vectors composed of any of $N-1$ components of the samples are linearly independent, the rank of matrix C is

$$r(C) = min(m, N-1) = M.$$

That is equation group (134) has solutions. Therefore, the following equation group

$$f_h(\sum_{j\neq 0} w_{j0}x_{ij}) = x_{i0} \qquad (i = 1, 2, ...M), \tag{135}$$

also has solutions. This is because a component of a sample can only be either + 1 or -1, therefore, $f(x_{j0}) = x_{j0}$. Equation (135) is obtained by applying the function to both sides of (134).

Consider those samples whose 0th component is +1 as one class and all other samples as another class. The above proof indicates that the two classes are linearly separable. According to the Perceptron convergence theorem [51, 62], at node 0, learning all the weights to node 0 using (133) is convergent. The same results can be obtained at other nodes. That is, all learning procedures for the weights of the whole network are convergent. We have

$$x_{ij} = f_h(\sum_{k\neq j} w_{kj}x_{ik}) \qquad (i = 1, 2, ...M, j = 1, 2, ..., N-1). \tag{136}$$

Equation (136) shows that all samples are stable attractors of the network.

Q. E. D.

If a pattern X is an attractor of the network, it is obvious that $-X$ is also an attractor of the network, because

$$f_h(-W^T X) = -f_h(W^T X) = -X,$$

where W is the weight matrix. Pattern $-X$ is called the complementary pattern of X. Therefore, the complementary of an attractor is also a stable attractor of the network. This can be avoided by adding a threshold for each node of the network.

Theorem 5.1 indicates that the number of the samples that can be stored in a network is almost the same as the number of nodes in the network. If the samples are well distributed in the space, the conditions in Theorem 5.1 can be satisfied easily and samples can be memorized by the network.

Although a Perceptron-based asymmetric network has twice as many independent coefficients as the Hopfield network, the number of the samples that can be stored in such a network with N nodes is approximately N. This algorithm can also distinguish very similar patterns. Certain problems that exist in the Hopfield model can be overcome in such a network. The learning procedure is very similar to human behavior in the sense that information is memorized through reciting over and over again. Fig. 23 shows ten binary 10×10 images to be stored. For the 10 sample images shown in Fig. 23, the learning algorithm converges quickly. All images can be stored by the network. The same network can also store the 36 images shown in Fig. 24.

5.3 Learning Pattern Sequence

The dynamic behaviors of a symmetric network are relatively simple: the system relaxes to states which are (local) minima of a global energy function. Once such a local minimum has been reached it remains absolutely stable for sequential evolution, or may move between two states for parallel evolution. Symmetric networks cannot

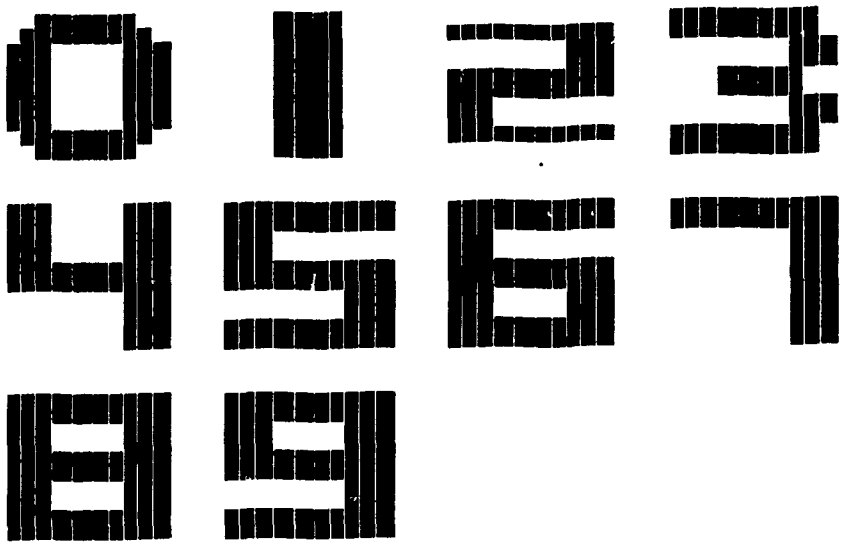

Figure 23: Ten 10×10 binary images to be memorized by an associative memory network

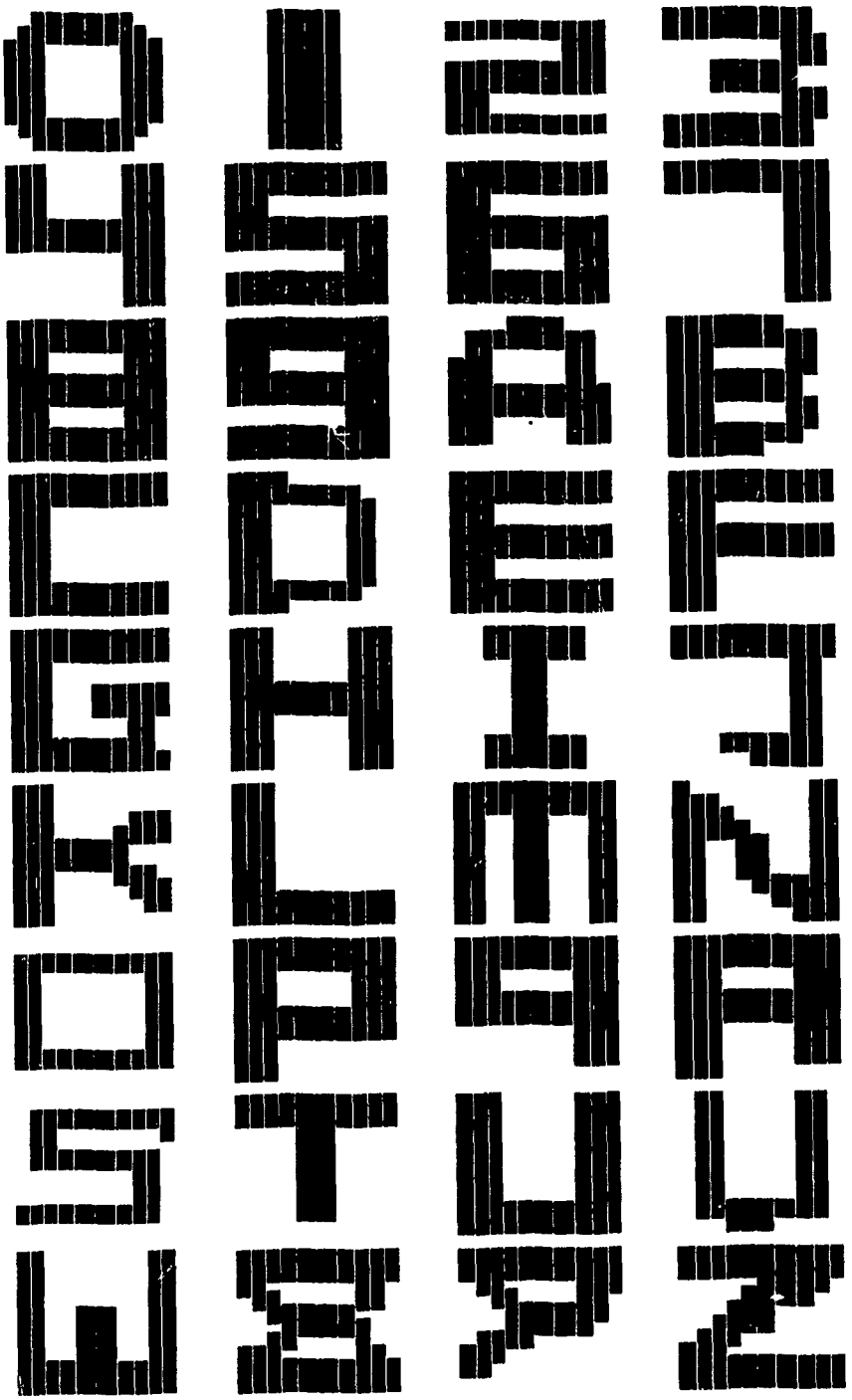

Figure 24: 36 10×10 binary images stored in an associative memory network

provide temporal association. They lack the ability to retrieve a sequence of patterns using a single recalling input. In paper [72], an asymmetric network is given for embedding a sequence of patterns. However, the weight of the network are also decided by the Hebbian rule as in the Hopfield model. Therefore, it has the same learning limitations as the Hopfield network.

The model proposed above is extended to embed one or more sequences of patterns, or cycles of patterns in the network. If the patterns to be learned are arranged as

$$X(0) \rightarrow X(1), X(1) \rightarrow X(2), ..., X(i) \rightarrow X(i+1), ..., X(M) \rightarrow X(M+1).$$

Then, for pattern $X(t), X(t+1)$, the learning algorithm (133) is rewritten as

$$w_{ij}(t+1) = w_{ij}(t) + \eta[x_j(t+1) - f_h(\sum_{k \neq j} w_{kj}(t)x_k(t))]x_i(t). \qquad (137)$$

If pattern $X(M+1) = X(0)$, a pattern cycle $X(0) \rightarrow X(1) \rightarrow ... \rightarrow X(M) \rightarrow X(0)$ is formed. Several cycles can be embedded in a network by arranging corresponding learning sequence. The memory capacity for a number of patterns in the pattern sequences is the same as that for stable pattern attractors.

5.4 A Method for Enhancement of Associative Memory

Experimental results explicate that although the samples can be made as the stable attractors of the network, when a large number of samples are stored in a network, many non sample attractors are also produced. Because the network is asymmetric, there is not a general energy function. Asymmetric nonlinear dynamic networks produce more complex phenomena than symmetric ones. Some behaviors of a dynamic

system may be useful. The others may be unwanted results. It is important to have some methods to control the dynamics of the system.

When an input pattern is not a sample, the associative result may be a non-sample attractor, which makes the recall fail. We can make the sample attractors somewhat different from non-sample attractors. Here an improved learning algorithm is given to achieve this.

The method is consistent with the psychological phenomenon of impression deepening. After some samples have been remembered, people often repeat them more times so that the impression of the samples is deepened. The impression deepening method enables a network to escape from non-sample attractors and to attain sample attractors. It is achieved by modifying the Perceptron learning algorithm as follows.

Let

$$s_j(t) = \sum_{k \neq j} w_{kj}(t) x_k(t).$$

$$w_{ij}(t+1) = \begin{cases} w_{ij} & \text{if } x_j = f_h(s_j(t)) \text{ and } |s_j(t)| > T, \\ w_{ij}(t) + \eta x_j(t) x_i(t) & \text{otherwise} \end{cases} \tag{138}$$

where $T > 0$ is called an impression threshold, and $f_h(s_j)$ is the hard limiter nonlinearity. The above learning algorithm requires not only that a sample be made an attractor of the network, but also the weighted sum of incoming signals to each node in the network must be larger than a given threshold T. However, for a non-sample attractor, such conditions are not guaranteed. According to this, when operating, the system can escape from non-sample attractors to sample attractors. The state evolution equation is given as follows.

Let the initial input pattern be $X(0)$ and the state of the system after k steps of learning be $X(k)$. Let

$$S_j = \sum_{i \neq j} w_{ij} x_i(k),$$

when $s_j \geq T$

$$x_j(k+1) = 1,$$

when $s_j \leq -T$

$$x_j(k+1) = -1,$$

When $-T < s_j < T$,

$$x_j(k+1) = \begin{cases} 1 & \text{with probability } (T+s_j)/2T, \\ -1 & \text{with probability } (T-s_j)/2T. \end{cases}$$

Therefore, when the network attains a sample attractor, its state no longer moves. If a nonsample attractor is reached, its state may continue moving. A nonsample attractor that is a stable state when $T = 0$ now becomes a non-stable state when $T > 0$. The randomness in setting $x_j(k+1)$ when $-T < s_j < T$ forces the network to escape from such a non-sample attractor.

This method is similar to the Simulated Annealing Method (SAM) [37, 30] in that some random disturbance is introduced to enable the network to escape from undesired attractors. The method also differs from the SAM in the following aspects. First, the aimed state for convergence for the SAM is only one, that is, the global minimum of an energy function. There may be many aimed convergence states for the impression deepening method. Furthermore, the impression deepening method does not require a cooling procedure that costs a lot of computation in the SAM. When an aimed state is reached, the state of the network no longer evolves with the impression deepening method, but it continues moving with the SAM. Finally, the SAM cannot decide whether or not a state is the aimed state. Use of other methods is needed to decide whether or not the associative results are the samples in a Hopfield network. For the impression deepening method, a final state of the network can be considered as one of the sample attractors with high confidence.

The samples to be stored are the patterns shown in Fig. 23 for the following experiments. For input samples with 10% noise, Fig. 25 shows the associative results using the the asymmetric network trained by (133). We can see that most of the output patterns are not the sample attractors. Using the network trained by the impression deepening equation (138), Fig. 26 depicts that all outputs of the network are the original samples for the inputs with 20% noise.

5.5 The Analysis of the Impression Threshold

The impression deepening method can improve the capability of an associative memory system. The learning algorithm is based on the Perceptron algorithm with a threshold T. A node in such a network can be considered as a sample Perceptron. Here it can be shown that the learning algorithm with a threshold T is a suboptimal linear classification. For a group of samples belonging to two classes A and B, each sample of class B is multiplied by -1. Let M be the maximum norm of the vectors. If the samples are linearly separable, then there is a weight vector W such that

$$W^T X_i > 0 \qquad (i = 1, 2, ..., n).$$

The learning algorithm is given as follows. The initial vector W_0 is set to a zero vector. After k steps of training, the weight vector is W_k. Then if

$$W_k^T X_k > T$$

the weights do not change. Otherwise,

$$W_{k+1} = W_k + X_k.$$

Let

$$\alpha = \max_{W} \min_{i \le n} \frac{W^T X_i}{||W||}.$$

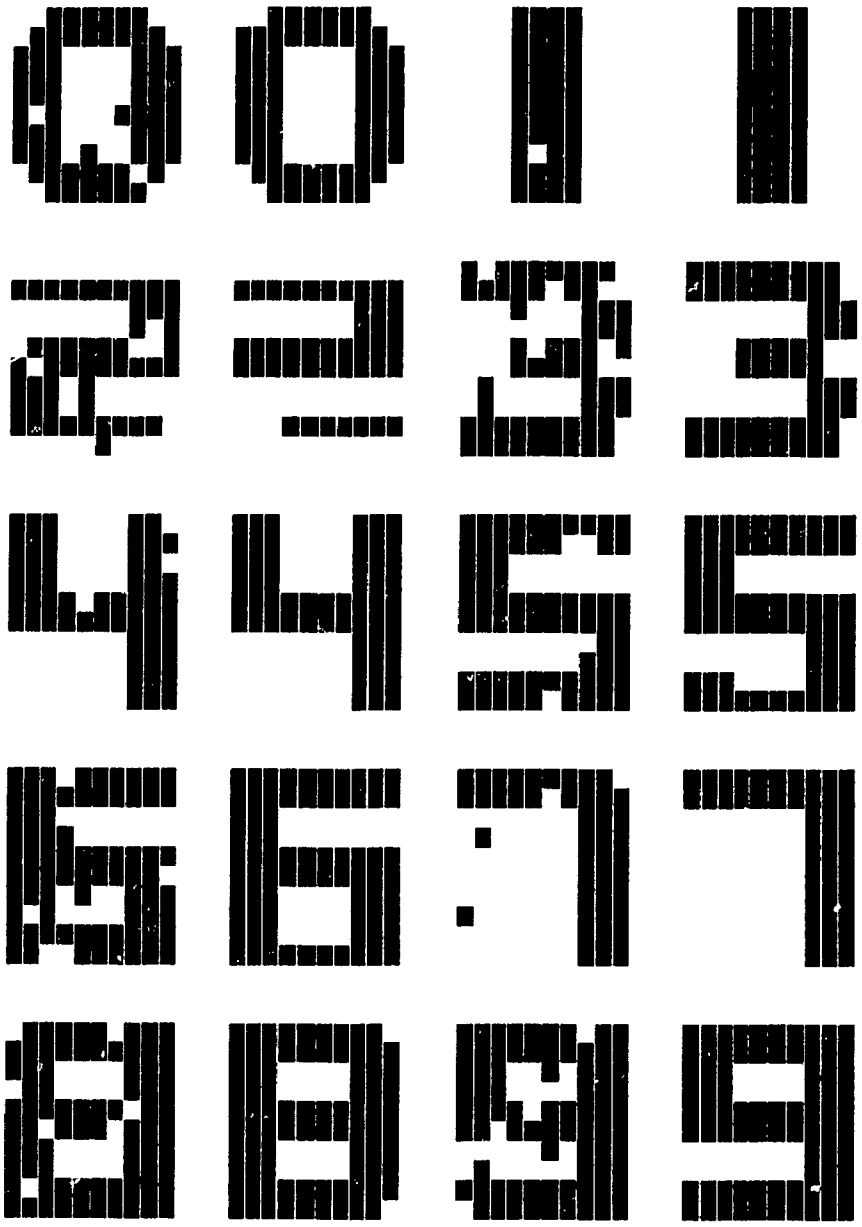

Figure 25: The associative recall results for inputs with 5% noise The first and the third column images are input patterns. The second and forth column images are the associated patterns.

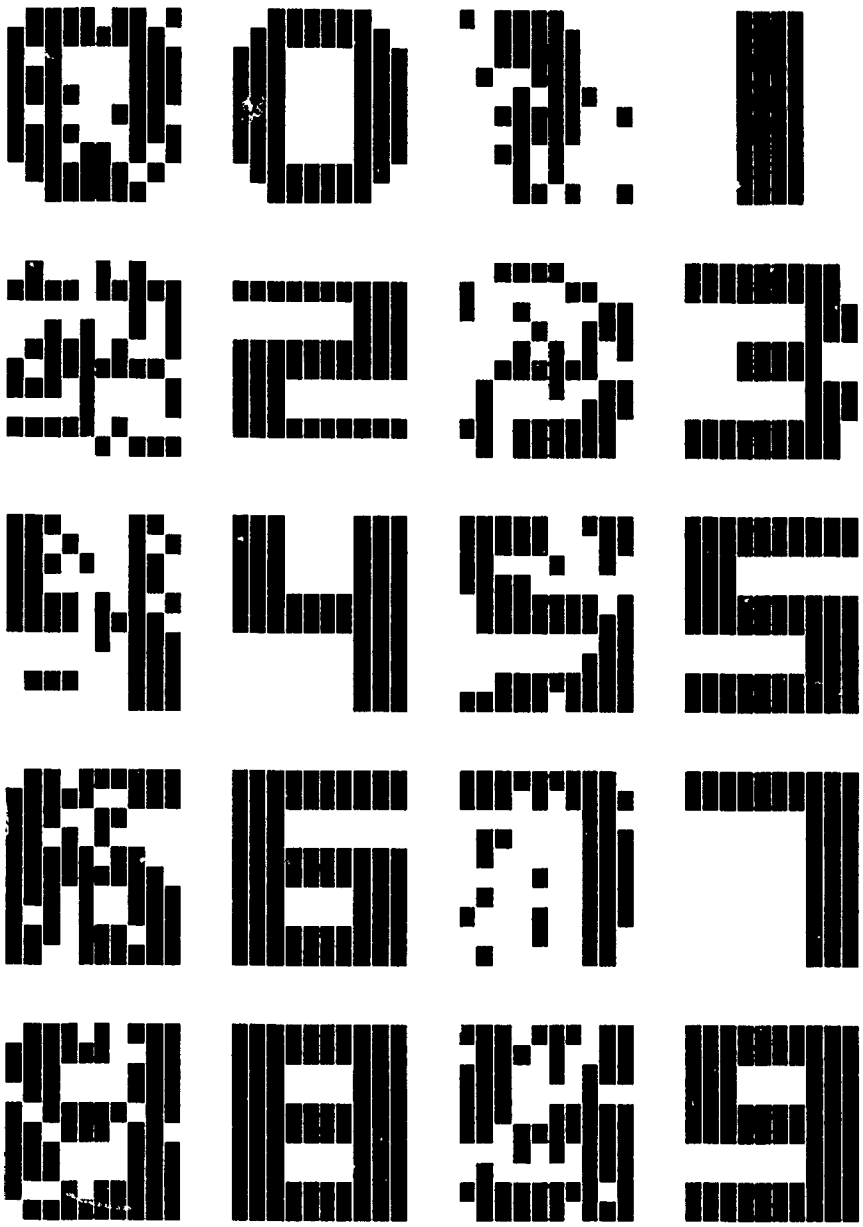

Figure 26: The recall results for inputs with 20% noise using the impression deepening method. The first and the third column images are input patterns. The second and forth column images are the associated patterns.

There exists a weight vector W_* with norm 1 such that $W_*^T X_i \geq \alpha$. The equation $W_*^T X = 0$ represents a hyperplane which is an optimal classification hyperplane in the sense that all samples are at least a distance away from the classification hyperplane $W_*^T X = 0$ in the extended pattern space. We proved the following theorem.

Theorem 5.2 *Assume that a group of samples are linearly separable in the sample space. When an impression threshold T is given, the learning algorithm is convergent. After convergence, for any sample X, we have*

$$\frac{W^T X}{||W||} \geq \frac{T}{2T + M}\alpha \tag{139}$$

where $M = \max_k ||Y_k||$.

Proof :

Let

$$G = \frac{W_*^T W_k}{||W_k||} . \tag{140}$$

It is obvious that $G \leq 1$. Let W_k be the sum of k samples. $W_k = Y_1 + Y_2 + ... + Y_k$, where Y_i is one of the samples. We have

$$W_*^T W_k = W_*^T (Y_1 + Y_2 + ... + Y_k) \geq k\alpha,$$

and

$$||W_k||^2 = ||W_{k-1} + Y_k||^2 = ||W_{k-1}||^2 + ||Y_k||^2 + 2W_{k-1}^T Y_k$$

$$\leq ||W_{k-1}||^2 + M + 2T \leq (M + 2T)k. \tag{141}$$

According to (140)

$$\frac{k\alpha}{\sqrt{(M+2T)k}} \leq 1. \tag{142}$$

That is, $k \leq \frac{M+2T}{\alpha^2}$. Therefore the learning algorithm is convergent. Substituting k into (141), we obtain

$$||W_k||^2 \leq \frac{(M+2T)^2}{\alpha^2}.$$

Therefore, for any sample X

$$\frac{W^T X}{||W||} \geq \frac{T}{2T+M}\alpha.$$

Q. E. D.

Theorem 5.2 shows that each sample point has a distance larger than $\frac{T}{2T+M}\alpha$ from the classification hyperplane in the sample space with an impression threshold. The minimum distance is related with the impression threshold T and α which is determined by the samples and does not change during learning. Therefore, increasing the threshold is the only way to enlarge the distance between the samples and classification hyperplane. For such a learning algorithm, if T is as large as the maximum norm of samples, the minimum classification distance is larger than $\frac{1}{3}\alpha$. The distance will approach $\frac{1}{2}\alpha$ when $T \to \infty$. When the algorithm is used for associative memory, the learning algorithm is suboptimal for the connections of each node. Therefore, each sample is constructed as a stable attractor by a group of suboptimal solutions. As a result, the associative capability of the system can be largely improved.

5.6 Learning algorithm for uncertain samples

In many real problems, some components of the samples are not important or unknown for classification. In some cases, a valid classification needs only a part of

the components of the samples. This implies that when only part of the features are present, a correct classification can be given no matter what the values of the other features are. Such a sample should not be represented by a point in space. Instead, it should be represented as a subspace. The features that are not in a sample are called uncertain components of the sample, the others are certain. The certain components of a sample are represented by 1 or -1. The uncertain components of sample can be represented by η , which may be either 1 or - 1. Therefore, a sample vector component can be 1, - 1 or η. To learn such samples, one approach is to map each sample to several samples with certain components before applying the Perceptron algorithm. However, the number of samples and learning time will increase quickly. Another method is direct learning for all samples through the following generalized learning algorithm.

For a set of samples, let o_k be the target output value of the kth sample that may be 1 or - 1. Let W_0 be an initial weight vector including threshold. W_0 can be arbitrarily chosen. After k training steps, assume that the weight vector is W_k. For sample X_k, let Ψ_k be the index set in which corresponding components of X_k are certain, and let Φ_k be the index set in which components of X_k are uncertain. Then, if

$$(\sum_{i\in\Psi} w_{ki}x_{ki} - \sum_{i\in\phi} |w_{ki}|)o_k \geq 0, \tag{143}$$

the weights are not changed and

$$W_{k+1} = W_k.$$

Otherwise, the weights are corrected as follows:

$$w_{k+1,i} = \begin{cases} w_{ki} + x_{ki}o_k & i \in \Psi_k, \\ w_{ki} - f_h(w_{ki}) & i \in \Phi_k. \end{cases} \tag{144}$$

When the weights no longer change for all samples, the learning ends. In this learning algorithm, if (143) is satisfied, the target output of x_k is same as the actual output no matter what values the uncertain components may assume. Note that different samples may have different components that are uncertain, i. e., Φ_k and Φ_m may be different for $m \neq k$. The union of uncertain components of all the samples may include every component of the whole sample vector. Therefore, we cannot simply set to zero all the connections to uncertain components.

The above algorithm may be interpreted as generating a training sample from the given sample. The actual training sample used is obtained by setting $x_{ki} = -f_h(w_{ki})$ for uncertain components. If (143) holds for this generated sample, the Perceptron will correctly classify the sample vector no matter what values the uncertain components may take.

From (144), it can be seen that the correcting algorithm is the same as the Perceptron algorithm for weights corresponding to certain components. For weights corresponding to the uncertain components, the absolute value of each weight is decreased by 1. It is easy to prove that the algorithm is convergent for linearly separable samples. Therefore, the Perceptron algorithm is easily generalized to include samples with uncertain components.

5.7 Summary

In this chapter a Perceptron-based asymmetric associative memory model has been proposed. The network is shown to have superior performance to the Hopfield symmetric connection model. The network is also able to store sequences and cycles of patterns that can be recalled by an input pattern. Under some conditions, the number of sample patterns that can be stored in the network is approximately the same

as the number of nodes in the network. It is shown that the impression deepening method is successful in enabling the network to escape from non-sample attractors and improve the associative performance greatly.

Chapter 6

Improved Back-Propagation Algorithm

6.1 Introduction

The Back-propagation (B-P) algorithm is the most commonly used neural network model [65, 66]. Back-propagation allows us to train the weights in a feedforward network of an arbitrary structure by following the gradient of steepest decent path in weight space, where the energy surface is usually defined by the mean squared error between desired and actual outputs of the network. There have been many examples of successful use of back-propagation for performing different tasks.

Unfortunately, back-propagation has some problems. First, the energy surface may have many local minima, so the algorithm can not always be guaranteed to converge to the optimal solution. The second problem is that it is difficult to analyze the behavior of hidden units in a multilayered network. Consequently it is not easy to estimate the exact number of the hidden units required for a given problem before the network is trained. The third problem is that the back-propagation algorithm is often slow.

The weights of the network after training depend on several factors. Among them

one may mention the randomly chosen initial weights and the sequence of training examples. The hidden units have approximately equal variance [2, 12]. For some problems, such as image coding and compression, these factors may reduce the usefulness of the B-P. This is due to the fact that the bits must be allocated evenly among the weights of the network and noise cannot be eliminated by removing the units with lowest variance. If the network is designed with too many hidden units then the additional error introduced is spread evenly throughout the units and cannot be easily detected or removed by looking at the signal to noise ratio of the individual units. Also, the generalization is usually poor for a network with a lot of redundant hidden units. On the other hand, if the network does not have enough hidden units, then the learning procedure may never converge. There are algorithms in the literature [43, 44] that can add or delete hidden units from the network. However, it is not easy to decide when and where the structure of the network should be changed. Since the variance of the hidden units is at the same level, a large error could be introduced by removing any hidden unit from the network.

In this chapter, we give an improved back-propagation (IB-P) algorithm. For a three layer network with one hidden layer, the initial variance of the weights and the learning rates are set differently for different layer units. The algorithm results in the hidden units having different variances. The learning algorithm can also reduce the possibility of getting trapped in local minima. In the conventional back-propagation algorithm all hidden layer units have almost the same degree of importance. The hidden layer units in the network trained by our method have different degrees of importance. A procedure for dynamically adjusting the network architecture is also proposed.

6.2 The Improved B-P algorithm

The back-propagation learning algorithm is summarized by the following equations. The forward propagation algorithm is defined as

$$net_{pj} = \sum_i w_{ji} o_{pi} \tag{145}$$

$$o_{pj} = f_j(net_{pj}). \tag{146}$$

The back-propagation algorithm is

$$\Delta_p w_{ji} = \eta \delta_{pj} o_{pi}. \tag{147}$$

The error signal is given by

$$\delta_{pj} = \begin{cases} (t_{pj} - o_{pj}) f_j'(net_{pj}) & \text{if the neuron is an output unit,} \\ f_j'(net_{pj}) \sum_k \delta_{pk} w_{kj} & \text{otherwise.} \end{cases} \tag{148}$$

The B-P algorithm is improved as follows: first, we set the initial weights randomly with different variances. For the weights that connect to the hidden unit k, the variance is set to $V_k = \alpha^k V_0$ where $0 < \alpha \leq 1$. The weights can be initialized by any random distribution. In (147), the learning rate η is constant. We also choose the learning rate to be different for different hidden layer units. For a three layer network with one hidden layer, equation (147) for hidden neuron k is changed to

$$\Delta_p w_{ji} = \eta_k \delta_{pj} o_{pi}. \tag{149}$$

The learning rate η_k is now given by

$$\eta_k = \beta^k \eta_0.$$

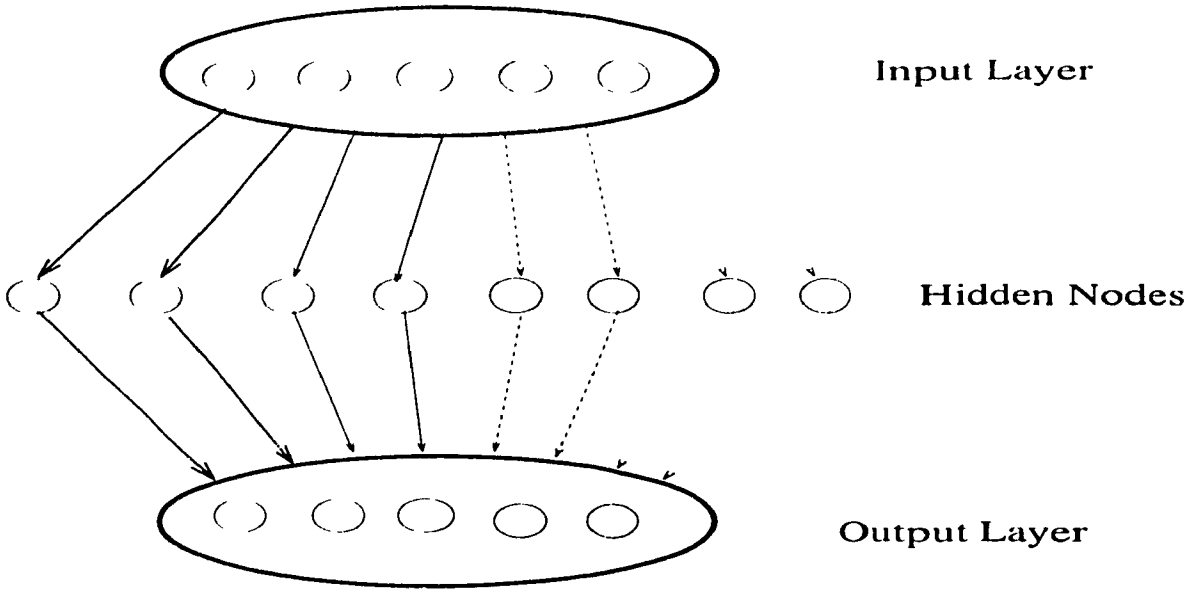

Figure 27: The initial neural network

where η_0 is a positive constant and $0 < \beta \leq 1$.

In this algorithm the weights that connect to the first hidden units are adjusted most, i.e. the first hidden neuron is the most important one. After the utility of the first hidden neuron is exhausted, the second neuron becomes more significant, etc. If for instance the network needs m hidden units for training, the hidden neurons after the mth one are adjusted with a very small variance. Therefore, these hidden neurons can be removed from the network without affecting the performance significantly. The last hidden neuron is the least important one.

Figures 27 and 28 show the change of a network before and after training. The thickness of the lines indicates the strengths of the connections. The hidden nodes with thin lines can be removed from the network in Figure 28 with little effect on the performance of the network.

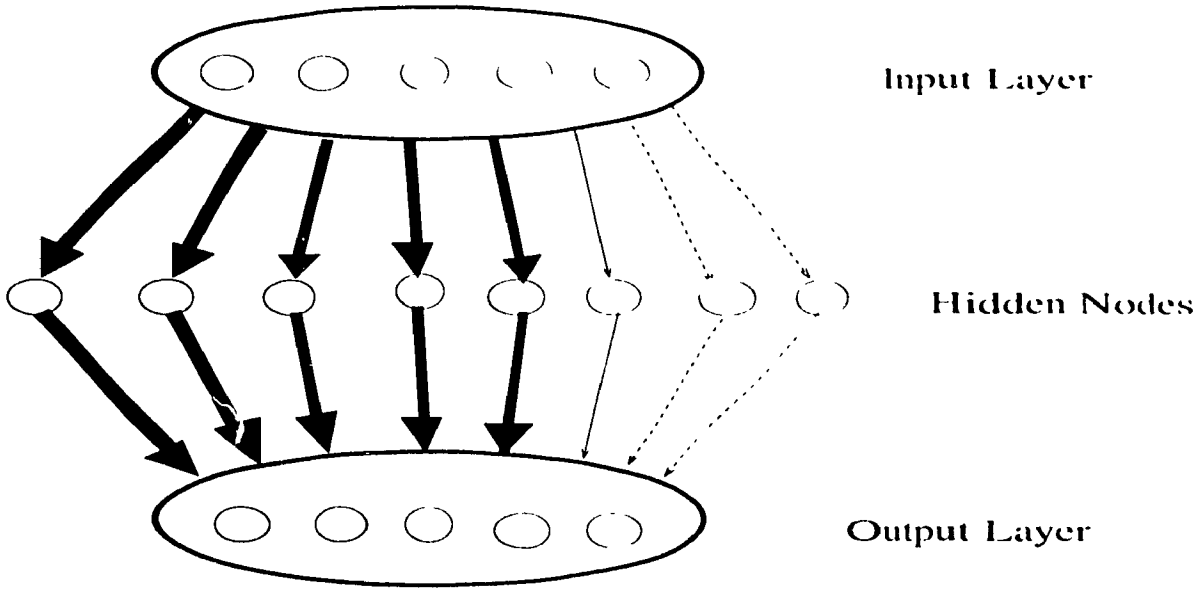

Figure 28: The trained neural network

6.2.1 IB-P algorithm analysis

The activation function for all the neurons is selected as

$$f(x) = \frac{2}{1+e^{x}} - 1.$$

Since $f'(x) = (1 + f(x))(1 - f(x))/2 \leq 1/2$, it is easy to prove that

$$|f(x)| \leq |x/2|$$

and

$$|f(x + \Delta x) - f(x)| \leq |\Delta x/2|.$$

For the ith hidden neuron, we have

$$|o_{pi}| = |f(net_{pi})| \leq |net_{pi}|/2 = |\sum_{j} w_{ji} o_{pj}|/2 \leq W_o I/2, \qquad (150)$$

where $W_o = \sum_j |w_{ji}|, I = max_j o_{pj}$. The input from neuron i to output neuron k is $w_{ik} o_{pi}$. If the hidden neuron i is removed, the change of the output in neuron k is

$$|\Delta o_{ik}| = |f(net_{pk}) - f(net_{pk} - w_{ik}o_{pi})| \leq |w_{ik}o_{pi}|/2$$

and

$$|\Delta o_{ik}| \leq |w_{ik}|W_o I/4 \leq (|w_{ik}| + W_o)^2 I/16 = S_{ik}^2 I/16, \tag{151}$$

where $S_{ik} = (W_o + |w_{ik}|)$ is the sum of the absolute value of the weights from the input neurons to the hidden neurons and from the hidden neurons to the output neurons. If $I = 1, |\Delta o_{ik}| \leq S_{ik}^2/16$. Therefore, if a hidden neuron is deleted from a trained network, the change of the output in neuron k depends on S_{ik}. If the network has only one output neuron, S_{ik} is written as S_i.

For pattern classification, a training set is correctly classified if the largest output error over the entire set is less than one. Let β designate the maximum error between the output and target output for the trained network over the entire sample space. The hidden neuron i is deleted from the network if $S_i < 4\sqrt{1-\beta}$. If the hidden neuron i is one of the neurons that can not be removed from the network, regardless of the learning rate and the initial weights, then we have $S_i \geq 4\sqrt{1-\beta}$.

6.3 An Adaptive Structure Neural Network

An adaptive structure neural network can be achieved through adjusting the number of neurons in the hidden layers dynamically. By using the IB-P learning algorithm, the hidden neurons have different variances. If a hidden neuron has small S_{ik} relative to the other neurons, it can be removed from the network dynamically. For pattern classification problems, if e_m defined as the maximum error between the desired output and the output of the network over the entire training set is less than 1, then the result is considered as correct classification. The maximum error introduced by removing neuron i from the network is

$$e_{ri} \leq S_i^2/16.$$

After node i is removed, e_{Ti}, the maximum difference between the desired output and network output, becomes

$$e_{Ti} \leq e_m + e_{ri} \leq e_m + S_i^2/16.$$

If $S_i < 4\sqrt{1-e_m}$, then $e_{ri} < e_m + (1-e_m) = 1$.

Therefore, if $S_i < 4\sqrt{1-e_m}$, node i can be removed and the network can still provide a correct classification.

For a function approximation problem, a positive number ν which is larger than the expected error is selected. If the actual error e_m of the network is less than ν and

$$S_i < 4\sqrt{\nu - e_m}/\sqrt{I},$$

we can remove the hidden neuron i from the network. The maximum error introduced by removing the hidden neuron i from the network is

$$e_{ri} \leq S_i^2 I/16 = \nu - e_r.$$

After neuron i is removed, the maximum output error is

$$e_{Ti} \leq e_m + e_{ri} \leq e_m + \nu - e_m = \nu.$$

So even if the hidden neuron i is removed the actual output error remains less than ν.

6.4 Experimental Results

6.4.1 XOR and the Partition problem

As a benchmark example we solve the XOR problem to test our algorithm. It is well known that a network with two hidden nodes can correctly solve the problem. To

train our network we start with 4 hidden units using the improved B-P algorithm with $\eta = 0.5$ and $\alpha = 5.0$. The following results were obtainedand are in Figure 29 which shows the changes of S_i. After the network is trained 5200 steps, we have $S_0 = 14.1, S_1 = 13.1, S_2 = 1.47$, and $S_3 = 0.038$. According to (151), if we remove hidden unit 2 we get $|\Delta o_2| < 1.47^2/16 = 0.14$. Similarly if we remove hidden unit 3 we get $|\Delta o_3| < 0.00006$. Therefore, hidden unit 3 contributes very little to the outputs of the network. Consequently the network gives a correct classification with hidden neuron 2 deleted.

The algorithm also reduces the probability of entering local minima. For the XOR problem, the probability of entering local minima is 10% and 3% for $\alpha = 2$ and $\alpha = 5$, respectively, whereas the probability of entering local minima is 33% for the original B-P algorithm.

For the parity problem with 3 inputs, if the network has 3 hidden neurons, we found that 4 out of 300 cases got trapped in local minima by using the original B-P algorithm with random initial weights. All 300 learning cases are convergent using improved B-P learning algorithm with $\alpha = 5$. If the network has 2 hidden neurons, the probability of entering local minima is 0.2 and 0.5 for the IB-P and the B-P methods, respectively. For the parity problem with 4 inputs and 5 hidden units after the network has been trained with $\alpha = 5.0$ and $\eta = 0.5$, the sum of the absolute values of the weights is given in Table 8 for eight learning procedures with different initial random weights. We can see that there is a large difference between the variance of the unremovable hidden units and the variance of the removable hidden units, while there is little difference among the variances of unremovable hidden units.

Fig. 30 shows the sum of absolute weights for a network with four hidden units using the IB-P learning algorithm.

Fig. 30 shows the change of the weights if we train a network with four hidden

case number	S_0	S_1	S_2	S_3	S_4
1	10.220725	7.223655	5.098019	0.294625	0.051301
2	9.609260	8.124382	5.240192	0.011271	0.011109
3	9.338788	7.360351	4.720688	0.092741	0.032782
4	9.338788	7.360351	1.720688	0.092741	0.032782
5	9.819942	8.174868	5.250357	0.132380	0.025526
6	12.271561	11.087083	7.780679	6.153959	0.140350
7	8.238710	7.184856	5.073183	0.089974	0.025867
8	10.716545	8.312218	5.125797	0.178817	0.046587

Table 8: The training results using the improved back-propagation algorithm for the parity problem

neurons and $\alpha = 5$ for a four input parity problem using the dynamic structure method. We can see that hidden neurons 3 and 4 are removed automatically within 100 steps.

For an unremovable hidden unit i, S_i must satisfy

$$S_i \geq 4\sqrt{1-\beta},$$

where β is the desired error. For example, with $\beta = 0.1$ for the learning results in Table 8, $S_i > 4\sqrt{0.9} = 3.79$ and we see that all the unremovable hidden units satisfy this condition.

Therefore, the sum of the absolute values of the weights of hidden units is in the same level. However, the sum of the absolute value of the weights of removable hidden units is much smaller.

6.4.2 Function Approximation

A neural network can also be used to approximate a function. As an example consider the function

$$f(x) = sin(2\pi x)cos(6\pi x)/3 + 2\pi x/9.$$

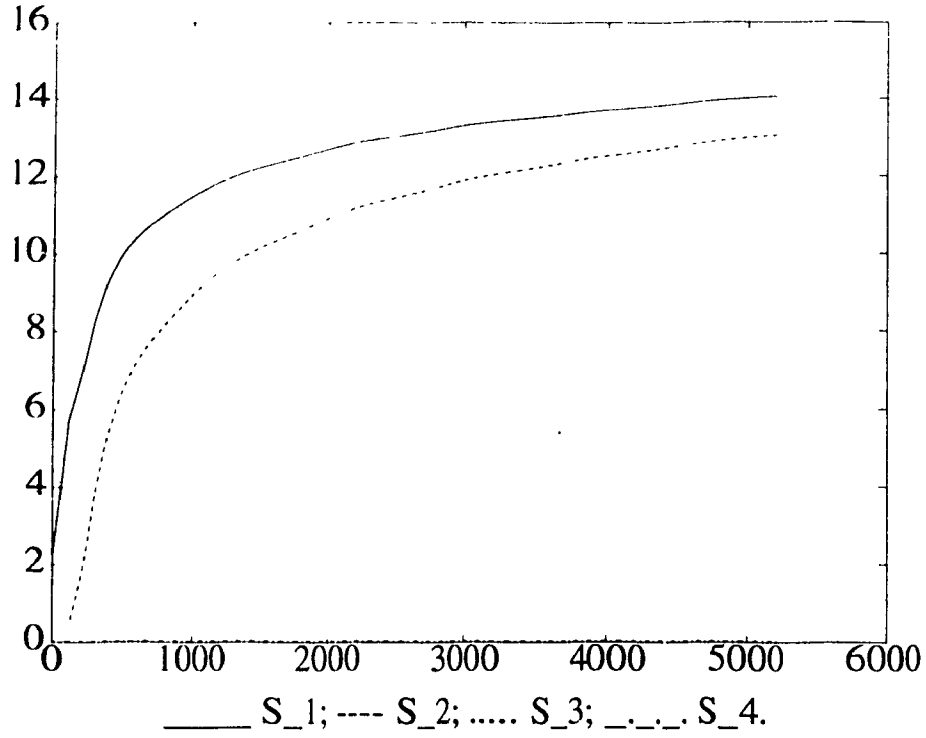

Figure 29: The sum of the absolute values of weights using the IB-P algorithm

When we use a network with 9 hidden neurons to approximate this function, after 45000 training steps with the B-P algorithm the output function of the network is shown in Figure 31. The sum of the absolute values of weights S_i after the network is trained is given by

$$S_0 = 31.47, S_1 = 3.56, S_2 = 5.15, S_3 = 18.11, S_4 = 4.98,$$
$$S_5 = 5.84, S_6 = 6.00, S_7 = 5.30, S_8 = 5.21$$

Figures 32(a) and (b) show the outputs of network with hidden node 3 and 5 removed, respectively. When the IB-P method is used to train the network with 9 hidden neurons, with $\eta = 0.5$ and $\alpha = 2.0$, the sum of the absolute values of the weights S_i after the network is trained is given by

$$S_0 = 8.11, S_1 = 18.79, S_2 = 27.54, S_3 = 5.79, S_4 = 1.07,$$

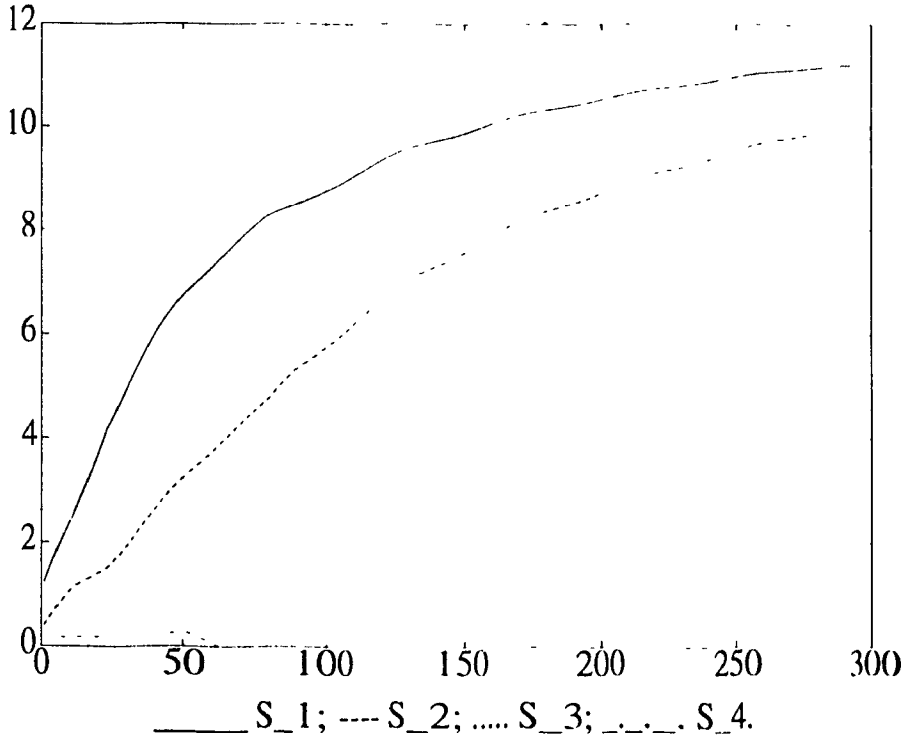

Figure 30: The sum of the absolute values of weights using the adaptive structure scheme

$$S_5 = 0.083, S_6 = 0.093, S_7 = 0.0092, S_8 = 0.0035.$$

We can see that the last 4 hidden units have contribute very little to the output and therefore can be deleted from the network. Figure 33(a) shows the response of the trained network and the desired output. Figure 33(b) shows the output of the network with the last four hidden units removed. If we use the adaptive structure algorithm that removes the hidden units dynamically we get the following results in Figure 34. The network selected has 8 hidden units at the beginning of training with ν set to 0.1. Figure 34 shows the change of the sum of absolute values of weights.

Figure 35(a) shows the network output before the structure is changed at about 44000 steps. Figure 35(b) is the result of the final network with 5 hidden neurons at the end of the training cycle.

6.5 Summary

The algorithm proposed in this chapter improves some problems of the back-propagation algorithm. After a network is trained, we clearly see which and how many hidden units are needed to solve the given problem. The network can also reduce the possibility of entering local minima. According to the analysis given in this chapter, we can remove the hidden neurons from the network both statically and dynamically. The algorithms proposed in the chapter are applicable to solve problems such as pattern recognition, function approximation, image compression, etc.

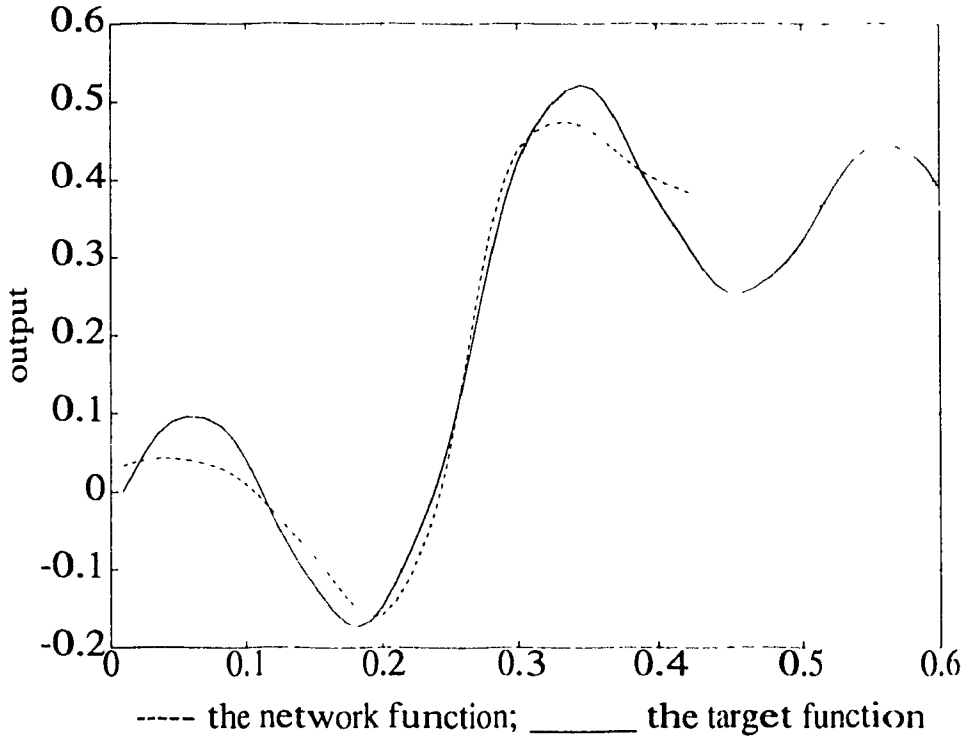

Figure 31: The output of the network using the B-P training algorithm

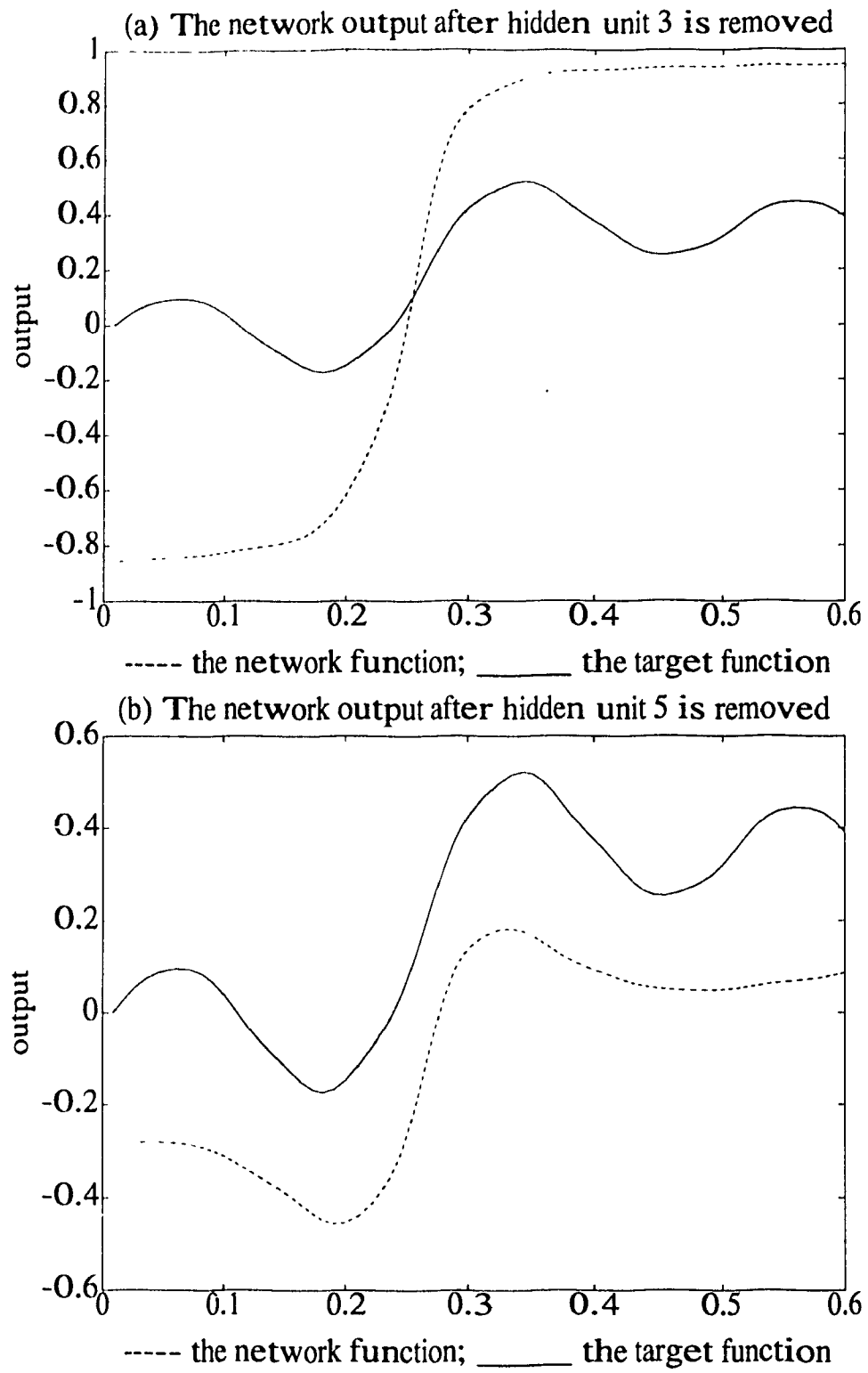

Figure 32: The output of the network using the B-P training algorithm after hidden neurons are removed

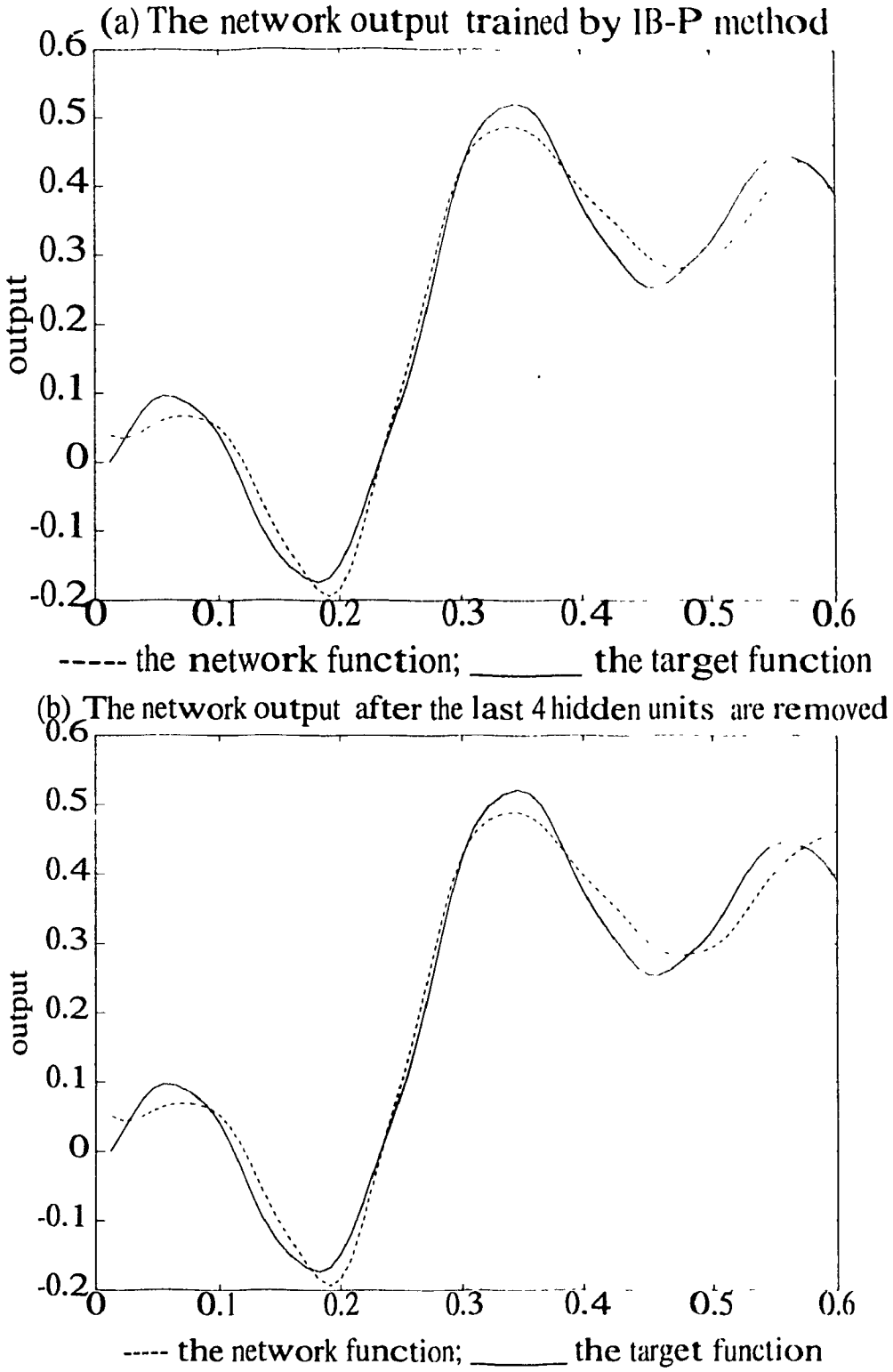

Figure 33: The output of the network using the IB-P training algorithm

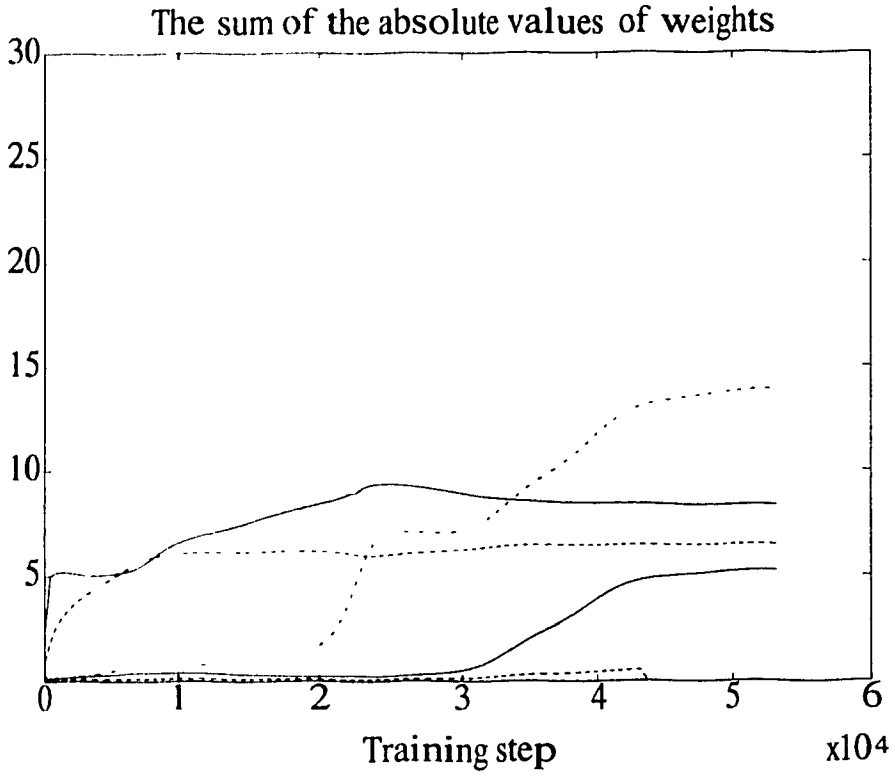

Figure 34: The strengths of the connections for the function approximation using the adaptive IB-P

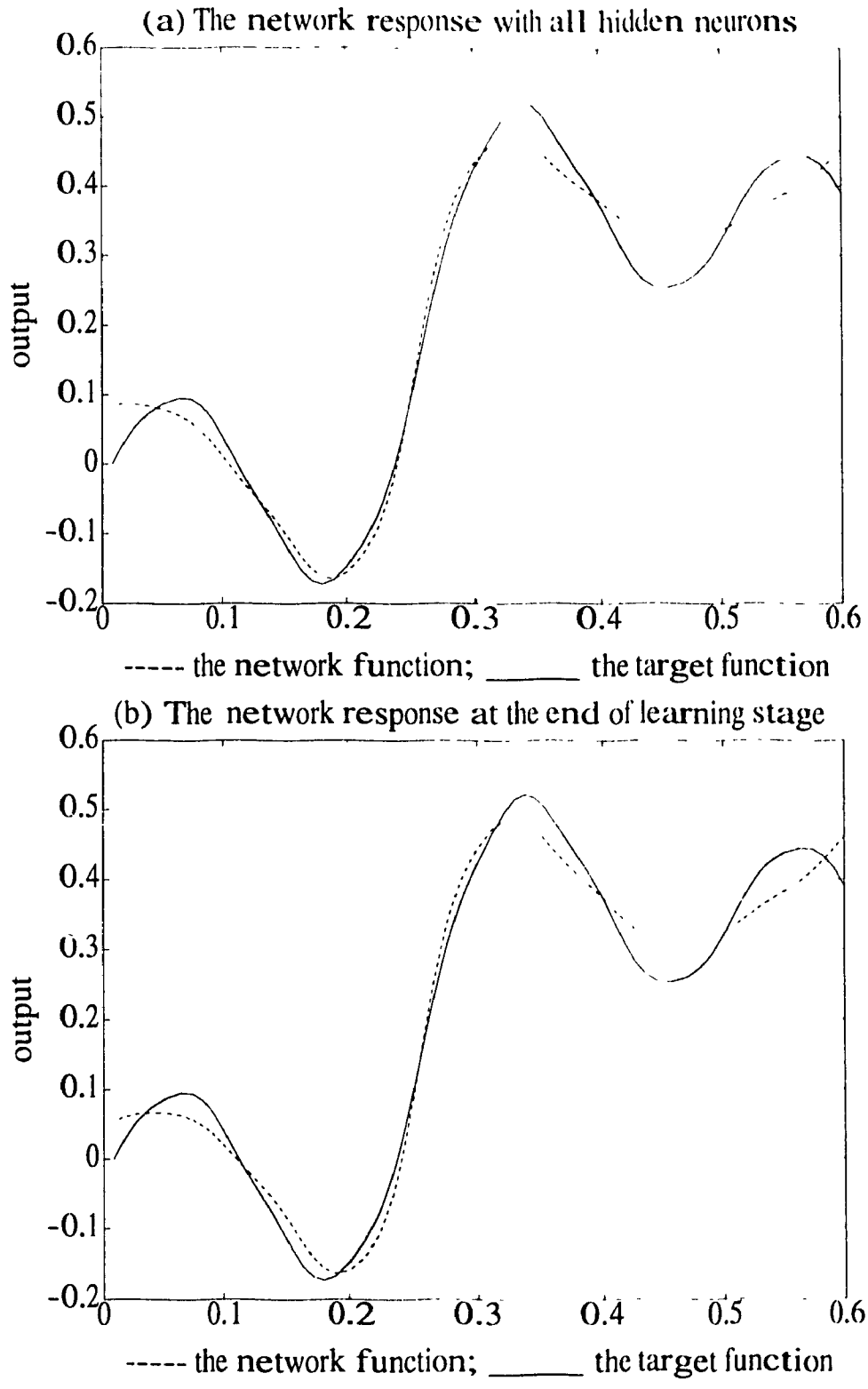

Figure 35: The function approximation using the adaptive structure scheme

Chapter 7

Conclusion

In this thesis, first, the unsupervised Perceptrons are proposed and the convergence of the algorithms is proven under some conditions. The algorithms have very simple forms and are easy to be implemented. Since the unsupervised Perceptrons correspond to a group of absolute error criteria, it is theoretically and experimentally shown that they are more robust than the Oja learning algorithms for principal component analysis when outliers exist in the learning samples.

The stochastic approximation theory is a very useful tool to analyze the convergence of neural network learning algorithms. The theory is extended to analyze the other important characteristics of learning algorithms in this thesis. The learning speed and robustness are formally defined and analyzed based on the stochastic approximation theory. It is shown that the learning speed and the robustness of a learning algorithm are highly related and mainly depend on the learning rate and the eigenvalues of the training samples. By selecting a large learning rate parameter, we can increase the learning speed of a learning algorithm. However, the robustness of the algorithm will be decreased correspondingly. The learning speed and the robustness provide a sort of global performance measure of a learning algorithm. The explicit learning speed and robustness formulas are derived for the unsupervised Perceptrons,

the Oja learning algorithms and the Widrow-Hoff learning algorithms.

The learning error, as another important measure of the local performance of a learning algorithm, is composed of the bias and the variance. An optimal learning rate is derived by minimizing the learning error. The bias and variance analysis make it possible to compare the estimation error of the traditional statistical algorithms and the learning error of neural network algorithms.

The binary tree classifier proposed in this thesis can be constructed very fast. The tree is also well balanced. At each node, it is shown that the hyperplane can minimize the partition error. When the classifier is used for Chinese character recognition, for 3000 characters, the recognition speed is up to 200 Chinese characters per second. Also, the tree classifier is successfully applied to waveform recognition. If the number of classes is not very large, the tree classifier can be trained by the Fisher's multiple linear discriminant method.

Another model proposed in the thesis is an asymmetric associative memory network which is trained by the Perceptron learning algorithm. Under some conditions, it is proven that the number of the samples that can be stored in such a network is the same as the number of nodes in the network. The impression deepening method is successful in enabling the network to escape from non-sample attractors and improve the associative performance greatly.

Finally, the B-P algorithm is improved by setting different learning rates and initial weight strengths for a network. After training, it is shown that there is a big gap between the weight values of useful hidden neurons and unimportant hidden neurons. It is proven that only a small error is introduced after pruning the unimportant neurons. Therefore, the algorithm can find the number of useful hidden neurons automatically. Also, the improved B-P algorithms can decrease the probability of the learning entering a local minimum.

7.1 Future work

The work presented in this thesis mainly focus on the theoretical analysis and algorithms. We will continue to give some detailed analysis. More attention will be paid to the applications of these algorithms in the future.

First, we will try to extend the unsupervised Perceptrons to find the first k eigenvectors. Also, the corresponding analysis should be given. The relationship between the unsupervised Perceptrons and other unsupervised algorithms will be explored, so that the theoretical analysis results on the unsupervised Perceptrons can be extended to other related algorithms and provide some guide for learning parameter selection.

Also, it is possible to compare the statistical learning algorithm and the corresponding neural network learning algorithm using the bias and variance analysis. For principal component analysis, some preliminary results suggest that neural network PCA approach requires less computation and less memory in comparison with the statistical PCA approach for high dimensional training data if an optimal learning rate is used. We still need more theoretical analysis and experimental results to compare these two kinds of learning algorithms.

The global analysis of the learning performances of the back-propagation algorithm is very difficult. But, the analysis given in this thesis may be applied to some local behaviors of the B-P algorithm.

For Chinese character recognition, we mainly focus on the construction of a good classifier and an efficient search algorithm in this thesis. There is also a lot of work to be done to develop a real OCR system. First, some efficient preprocessing algorithm should be given, such as document processing, line and character segmentation, etc. Then, different features will be selected and tested. We will pay more attention to the structure features of the characters. Since the training algorithm of the classifier

is very fast, it is possible to construct an adaptive dynamic OCR system which can be improved according to the recognition environment. Also, we need to treat the multiple fonts and sizes in the real application. The tree classifier given in this thesis can also be applied to hand written (both on line and off line) character recognition.

More simulations and analyses for the improved B-P algorithm are required for large scale problems. We will explore why the improved B-P algorithm does not easily enter local minima. For training a network with the same inputs and outputs using the improved B-P algorithm, we need to analyze the relationship between the kth hidden neuron and the kth principal component.

References

[1] P. Argentiero, R. Chin and P. Beaudet, "An automated approach to the design of decision tree classifiers", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-4, pp. 51–57, 1982.

[2] P. Baldi and K. Hornik, "Neural networks and principal component analysis: Le..rning from example without local minima", *Neural Networks*, vol. 2, pp. 53–58, 1989.

[3] E. B. Baum, "The perceptron algorithm is fast for nonmalicious distributions", *Neural Computation*, vol. 2, pp. 248–260, 1990.

[4] C. C. Blaydon, "Recursive algorithms for pattern classification", Office Naval Research Technical Report 520, Division of Engineering and Applied Physics, Harvard University, Cambridge, MA, 1967.

[5] A. Blum and R. L. Rivest, "Training a 3-node neural net is NP-complete", in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 1, pp. 494–501. Morgan Kaufmann, San Mateo, CA, 1989.

[6] D. G. Bounds, P. J. Lloyd, B. Mathew and G. Waddell, "A multilayer perceptron network for the diagnosis of low back pain", in *Proceedings of the 2nd IEEE International Conference on Neural Networks*, San Diego, CA, vol. 2, pp. 481–489, July 1988.

[7] L. Breiman, J. H. Friedman, R. A. Olshen and C. J. Stone, *Classification and Regression Trees*, Wadsworth, Belmont, CA, 1984.

[8] G. A. Carpenter and S. Grossberg, "A massively parallel architecture for a self-organizing neural network recognition machine", *Computer Vision, Graphics and Image Processing*, vol. 37, pp. 54–415, 1983.

[9] R. C. Casey and G. Nagy, "Decision tree design using a probabilistic model", *IEEE Transactions on Information Theory*, vol. IT-30, pp. 93–99, 1984.

[10] P. A. Chou and R. M. Gray, "On decision trees for pattern recognition", in *Proceedings of IEEE International Symposium on Information Theory*, Ann Arbor, MI, p. 69, 1986.

[11] P. A. Chou, T. Lookabaugh and R. M. Gray, "Optimal pruning with applications to tree structured source coding and modeling", *IEEE Transactions on Information Theory*, vol. IT-35, pp. 299-315, 1989.

[12] G. W. Cottrell, P. Munro and D. Zipser, "Learning internal presentation from gray-scale images: An example of extensional programming", in *Proceedings of the 9th Annual Conference of the Cognitive Science Society*, Seattle, WA, pp. 461–473, July, 1987.

[13] F. Critchley, "Influence in principal component analysis", *Biometrika*, vol. 76, pp. 354–362, 1981.

[14] S. J. Devlin, R. Gnanadesikan and J. R. Ketternrig, "Robust estimation of dispersion matrices and principal components", *Journal of American Statistical Association*, vol. 76, pp. 354–362, 1981.

[15] G. R. Dattatreya and L. N. Kanal, "Decision trees in pattern recognition", in *Progress in Pattern Recognition 2*, L. N. Kanal and A. Rosenfeld, Eds., pp. 189–239. Elsevier Science, New York, 1985.

[16] J. L. Doob, *Stochastic Processes*, Wiley, New York, 1953.

[17] R. O. Duda and P. E. Hart, *Pattern Recognition and Scene Analysis*, Wiley, New York, 1973.

[18] J. A. Feldman and D. H. Ballard, "Connectionist models and their properties", *Cognitive Science*, vol. 6, pp. 205–254, 1982.

[19] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position", *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[20] S. I. Gallant, "Connectionist expert system", *Communications of the ACM*, vol. 31, no. 2, pp. 152–169, Feb. 1988.

[21] S. I. Gallant, "Perceptron-based learning algorithms", *IEEE Transactions on Neural Networks*, vol. 1, no. 2, pp. 179–191, June 1990.

[22] S. B. Gelfand, C. S. Ravishankar and E. J. Delp, "An iterative growing and pruning algorithm for classification tree design", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-13, pp. 163–174, 1991.

[23] S. Geman, E. Bienenstock and R. Doursat, "Neural network and the bias variance dilemma", *Neural Computation*, vol. 4, pp. 1–58, 1992.

[24] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, pp. 721–741, 1984.

[25] S. Grossberg, "Competitive learning: From iterative activation to adaptive resonance", in *Neural Networks and Natural Intelligence*, S. Grossberg, Ed., pp. 213–250. MIT Press, Cambridge, MA, 1988.

[26] R. Hadley, "Connectionist. rule following, and symbolic manipulation", in *Proceedings of the Eighth National Conference on Artificial Intelligence*, Menlo Park, CA, pp. 579–586, July 1990.

[27] L. H. Hall and S.G. Romaniuk, "A hybrid connectionist, symbolic learning system", in *Proceedings of the Eighth National Conference on Artificial Intelligence*, Menlo Park, CA, pp. 783–788, July 1990.

[28] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan, New York, 1994.

[29] R. Hecht-Nielsen, *Neurocomputing*, Addison-Wesley, Redwood City, CA, 1990.

[30] G. E. Hinton, T. J. Sejnowski and D. H. Ackley, "Boltzmann machines: constraint satisfaction networks that learn", Technical Report CMU-CS-84-119, Carnegie-Mellon University, 1984.

[31] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons", in *Proceedings of the National Academy of Science*, Washington D. C., pp. 3088–3092, May 1984.

[32] J. J. Hopfield and D. W. Tank, "'Neural' computation of decisions in optimization problems", *Biological Cybernetics*, vol. 52, no. 3, pp. 141–152, 1985.

[33] P. J. Huber, *Robust Statistics*, Wiley, New York, 1981.

[34] I. Kanter and H. Sompolinsky, "Associative recall of memory without errors", *Physical Review A*, vol. 35, no. 1, pp. 380–392, 1987.

[35] J. Karhunen and J. Joutsensalo, "Learning of robust principal component subspace", in *Proceedings of IJCNN'93: International Joint Conference on Neural Networks*, Nagoya, Japan, pp. 2409–2412, Oct. 1993.

[36] J. Karhunen and J. Joutsensalo, "Nonlinear generalizations of principal component learning algorithms", in *Proceedings of IJCNN'93: International Joint Conference on Neural Networks*, Nagoya, Japan, pp. 2599–2602, Oct. 1993.

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

[38] S. Klasa, H. F. Yin and T. Li, "A tree classifier based on principal axis division", *Congressus Numerantium*, vol. 95, pp. 142–152, 1993.

[39] T. Kohonen, "Self-organized formation of topologically correct feature maps", *Biological Cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.

[40] T. Kohonen, *Self-organization and Associative Memory*, Springer-Verlag, Berlin, 1989.

[41] S. Y. Kung and C. I. Diamantaras, "A neural network learning algorithm for adaptive principal component extraction (apex)", in *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*, Albuquerque, NM, vol. 2, pp. 861–864, 1990.

[42] H. J. Kushner and D. S. Clark, *Stochastic Approximation Methods for Constrained and Unconstrained Systems*, Springer-Verlag, New York, 1978.

[43] T. C. Lee and A. M. Peterson, "Span: A neural network that grows", in *Proceedings of IJCNN'89: International Joint Conference on Neural Networks*, Washington, D. C., vol. 2, pp. 635, June 1989.

[44] T. C. Lee, *Structure Level Adaptation for Artificial Neural Network*, Kluwer, Boston, 1991.

[45] R. R. Lippmann, "An introduction to computing with neural nets", *IEEE Acoustics, Speech and Signal Processing Magazine*, vol. 4, no. 2, pp. 4-22, 1987.

[46] L. Ljung, "Analysis of recursive stochastic algorithms", *IEEE Transactions on Automatic Control*, vol. AC-22, no. 4, pp. 551-575, 1977.

[47] Z. P. Lo, M. Fujita and B. Bavarian, "Analysis of neighborhood interaction in Kohonen neural networks", in *Proceedings of the 5th International Parallel Processing Symposium*, Anaheim, CA, pp. 247-249, 1991.

[48] M. Marchand, M. Golea and P. Rujan, "A convergence theorem for sequential learning in two-layer perceptrons", *Europhysics Letter*, vol. 11, no. 6, pp. 487-492, 1990.

[49] J. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity", *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115-133, 1943.

[50] M. Mézard and J. P. Nadal, "Learning in feedforward layered networks: the tiling algorithm", *Journal of Physics A: Mathematical and General*, vol. 22, no. 12, pp. 2191-2203, 1989.

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

[52] N. J. Nilsson, *The Mathematical Foundations of Learning Machines*, Morgan Kaufmann, San Mateo, CA, 1990.

[53] E. Oja, "A simplified neuron model as a principal component analyzer", *Journal of Mathematical Biology*, vol. 16, pp. 267-273, 1982.

[54] E. Oja, "Neural computing and pattern recognition", *Tutorial 4, IJCNN'91: International Joint Conference on Neural Networks*, Singapore, Nov. 1991.

[55] E. Oja, *Subspace Methods of Pattern Recognition*, Research Studies Press, New York, 1983.

[56] E. Oja and J. Karhunen, "On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix", *Journal of Mathematical Analysis and Applications*, vol. 106, no. 1, pp. 69–84, 1985.

[57] E. Oja, H. Ogawa and J. Wangviwattana, "Learning in nonlinear constrained hebbian network", in *Artificial Neural Networks*, T. Kohonen, Ed., pp. 385–390. North Holland, 1991.

[58] Y. H. Pao, *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, Reading, MA, 1989.

[59] L. Personnaz, I. Guyon and G. Dreyfus, "Information storage and retrieval in spin-glass like neural networks", *Journal de Physique, Lettres*, Orsay, France, vol. 46, pp. 359–365, no. 8, 1985.

[60] J. R. Quinlan, "Induction of decision tree", *Machine Learning*, vol. 1, pp. 81-106, 1986.

[61] H. Robbins and S. Monro, "A stochastic approximation methods", *Annals of Mathematical Statistics*, vol. 22, pp. 400–407, 1951.

[62] F. Rosenblatt, *Principles of Neurodynamics—Perceptrons and the Theory of Brain Mechanisms*, Spartan Books, Washington D. C., 1962.

[63] F. Rosenblatt, "On the convergence of reinforcement procedure in simple perceptrons", Technical Report VG-1196-G-4, Cornell Aeronautical Laboratory, Buffalo, New York, 1960.

[64] Y. A. Rozanov, *Probability Theory: a Concise Course*, Dover, New York, 1969.

[65] D. E. Rumelhart and J. L. McClelland, Eds., *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, MIT Press, Cambridge, MA, 1986.

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

[67] D. E. Rumelhart and D. Zipser, "Feature discovery by competitive learning", *Cognitive Science*, vol. 9, pp. 75–112, 1985.

[68] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology", *IEEE Transaction on Systems, Man and Cybernetics*, vol. SMC-21, no. 3, pp. 660–674, 1991.

[69] T. D. Sanger, "Optimal unsupervised learning in a single layer linear feedforward neural network", *Neural Networks*, vol. 2, no. 6, pp. 459–473, 1989.

[70] T. D. Sanger, "Analysis of the two-dimensional receptive fields learned by the generalized Hebbian algorithm in response to random input", *Biological Cybernetics*, vol. 63, no. 3, pp. 221–228, 1990.

[71] R. R. Sokal and P. H. A. Sneath, *Numerical Taxonomy*, W.H. Freeman, San Francisco and London, 1963.

[72] H. Sompolinsky and I. Kanter, "Temporal association in asymmetric neural networks", *Physical Review Letters*, vol. 57, no. 22, pp. 2861–2864, 1986.

[73] D. W. Tank and J. J. Hopfield, "Simple 'neural' optimization network: An a/d converter, signal decision circuit and a linear programming circuit", *IEEE Transactions on Circuits Systems*, vol. CAS-33, pp. 533–541, 1986.

[74] J. T. Tou and R. C. Gonzalez, *Pattern Recognition Principle*, Addison-Wesley, Reading, MA, 1974.

[75] D. S. Touretzky and G. E. Hinton, "A distributed connectionist production system", *Cognitive Science*, vol. 12, pp. 423–466, 1988.

[76] G. G. Towell, J. W. Shavlik and M. O. Noordewier, "Refinement of approximate domain theories by knowledge-based neural networks", in *Proceedings of the Eighth National Conference on Artificial Intelligence*, Menlo Park, CA, pp. 861–866, July 1991.

[77] C. von der Malsburg, "Self-organization of orientation sensitive cells in the striate cortex", *Kybernetik*, vol. 14, pp. 85–100, 1973.

[78] M. T. Wasan, *Stochastic Approximation*, Cambridge University Press, Cambridge, 1969.

[79] B. Widrow, "Generalization and information storage on networks of adaline 'neuron' " , in *Self-organizing system*, M. C. Yovitz and G. T. Jacobi, Eds., pp. 435–461. Spartan Books, Washington, D. C., 1962.

[80] B. Widrow and M. E. Hoff, "Adaptive switching circuits", in *IRE Western Electronic Show and Convention*, Convention record, part 4, pp. 96–104, August 1960.

[81] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1985.

[82] L. Xu and A. Yuille, "Robust principal component analysis by self-organizing rules based on statistical physics approach", *IEEE Transaction on Neural Networks*, vol. 6, no. 1, pp. 131–143, January 1995.

[83] L. Xu, S. Klasa and A. Yuille, "Recent advance on techniques of static feedforward networks with supervised learning", *International Journal of Neural Systems*, vol. 3, no. 3, pp. 253–290, 1992.

[84] W. Y. Yan, U. Helmke and J. B. Mooe, "Global analysis of Oja's flow for neural networks", *IEEE Transactions on Neural Networks*, vol. 5, no. 5, pp. 674–683, Sept. 1994.

[85] H. F. Yin and J. W. Tai, "A new kind of associative network model", in *Proceedings of IJCNN: International Joint Conference on Neural Network*, Washington D. C., pp. 499–503, Jan. 1990.

[86] H. F. Yin and J. W. Tai, "An associative memory model of language", in *Proceedings of IJCNN: International Joint Conference on Neural Network*, San Diego, vol. 3, pp. 663–670, June 1990.

[87] H. F. Yin and P. Liang, "A connectionist expert system combining production system and associative memory", *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 5, no. 4, pp. 523–544, 1991.

[88] H. F. Yin, S. Klasa and T. Li, "The evolution of neural network structure", in *Workshop of PRICAI: Pacific Rim International Conference on Artificial Intelligence*, Beijing, China, pp. 120–130, Aug. 1994.

[89] H. F. Yin and S. Klasa, "Linear neural network learning algorithm analysis", in *Proceedings of ICNN'95: IEEE International Conference on Neural Networks*, Perth, Australia, vol. 5, pp. 2847–2852, Nov. 1995.

[90] H. F. Yin and S. Klasa, "Learning rate analysis of linear learning algorithms", in *Proceedings of ICNN'95: IEEE International Conference on Neural Networks*, Perth, Australia, vol. 5, pp. 2835–2840, Nov. 1995.

[91] H. F. Yin and S. Klasa, "Learning speed and robustness analysis of unsupervised learning algorithm", in *Proceedings of ICANN'95: International Conference on Artificial Neural Networks*, Paris, France, pp. 205–210, Oct. 1995.

[92] K. Yoshida, Y. Hayashi and A. Imura, "A connectionist expert system for diagnosing hepatobiliary disorder", in *Proceedings of MEDINFO'89: Sixth Conference on Medical Informatics*, Beijing, China, pp. 116–120, Oct. 1989.

Appendix

Stochastic Approximation Algorithms

The asymptotic behaviors of stochastic differential and difference equations can be analyzed using the theory of stochastic approximation. Kushner and Clark [42] proposed a typical iterative algorithm with the learning difference equation (LDE) in the following form:

$$W_{k+1} = W_k + \gamma_k(h(W_k) + A_k + D_k), \tag{152}$$

where $\{W_k\}$ is a sequence of vectors in $\mathbb{R}^n$; h is a continuous vector valued function on $\mathbb{R}^n$; $\{A_k\}$ and $\{D_k\}$ are random vector sequences to be defined presently and γ_k is a scalar parameter. Assume that

i) h is a continuous $\mathbb{R}^n$ valued function on $\mathbb{R}^n$.

ii) $\{A_k\}$ is a bounded (almost surely) sequence of $\mathbb{R}^n$ valued random variables such that $A_k \to 0$ almost surely.

iii) $\{\gamma_k\}$ is a sequence of positive real numbers such that

$\gamma_k \to 0$, $\sum_k \gamma_k = \infty$.

iv) $\{D_k\}$ is a sequence of $\mathbb{R}^n$ valued random variables such that for each $\epsilon > 0$,

$$\lim_{k\to\infty} P(\sup_{m\geq k} ||\sum_{i=k}^{m} \gamma_i D_i|| \geq \epsilon) = 0.$$

Under these assumptions, Kushner and Clark proved the following theorem for algorithm (152) (Theorem 2.3.1. in [42]):

Theorem .1 *Assume i) to iv) above for algorithm (152). Moreover, assume that W_k is almost surely bounded for all k. Let S be a locally asymptotically stable (in the sense of Liapunov) set to the ordinary differential equation (ODE)*

$$\frac{dZ}{dt} = h(Z),$$

with domain of attraction $D(S)$. Then the following holds almost surely: if there is a compact subset $A \subset D(S)$ such that $W_k \in A$ infinitely often, we have $W_k \to S$ as $k \to \infty$ almost surely.

Remarks: If the assumptions iii) and iv) in Theorem .1 are modified as

iii) $\gamma_k > 0$, $\gamma_k \to 0$ and

$$\sum_{k=1}^{\infty} \gamma_k = \infty, \quad \sum_{k=1}^{\infty} \gamma_k^2 < \infty, \tag{153}$$

iv) $E(D_k) = 0$ and $E(||D_k||^2)$ is bounded.

Then, $\{\sum_{i=k}^{m} \gamma_i D_i\}$ is a martingale sequence. There is an important martingale inequality of Doob [16] which states that

$$P(\sup_{m\geq 0} |G_m| \geq \epsilon) \leq \lim_{m} E|G_m|^2/\epsilon^2.$$

Applying this to our problem yields

$$P(\sup_{m\geq n} |\sum_{i=k}^{m} \gamma_i D_i| \geq \epsilon) \leq constant \sum_{i=k}^{\infty} \gamma_i^2/\epsilon^2,$$

whose right side goes to zero as $k \to \infty$. Therefore, the assumption iv) of Theorem .1 holds. Therefore, we can also use the modified assumptions iii) and iv) in our proofs.