DYNAMIC CONSTRUCTION OF BACK-PROPAGATION ARTIFICIAL NEURAL NETWORKS

A Thesis

Submitted to

The Department of Electronic Engineering

of

The Chinese University of Hong Kong

In

Partial Fulfilment of the Requirements

for the Degree of

Master of Philosophy

By Korris Fu-lai Chung

June, 1991

ABSTRACT

In recent years there has been considerable interest in artificial neural network (ANN) research, and it has exploded with impressive successes across a wide variety of applications. Among various ANN models, the Back-propagation (BP) network may well be recognized as the most widely-used one. However, BP network is far from perfect or even perfectly understood. One difficulty in adopting this model is the need to pre-determine a suitable network size, particularly, the number of hidden nodes. If it was under-estimated, the problem would not be solved. On the other hand, oversized network also suffers from inefficient hardware realization and degraded generalization performance, i.e., the ability to produce correct response when presented with unseen patterns. In this thesis, we address this issue by proposing a hybrid network construction algorithm which will self-determine an appropriate size for BP networks.

The proposed algorithm composes of two parts; they are network growth and network pruning. For the first part, a progressive training is used to construct a reasonably large network by adding hidden nodes one by one to an initially small network as the available training data become more and more complex. The constructed network then undergoes a node pruning process, the second part of the proposed algorithm. Four categories of excessive nodes identified from a study on the characteristics of hidden nodes in oversized networks are detected and pruned from the network. After pruning, the network is retrained to obtain the finalized network which is expected to be an optimal or nearly optimal one. With respect to previous works, the proposed algorithm has advocated a dynamic way to construct the required network and successfully taken use of the advantages from network growth and network pruning, which are two common approaches to obtain a suitable network size, to achieve fully automatic design of BP networks. The effectiveness of the proposed algorithm has been demonstrated through various experiments.

ACKNOWLEDGEMENTS

It is a pleasure to express my sincere thanks to my supervisor, Dr. T. Lee, who spent so much effort to shape me in various aspects throughout the course of this research. I am deeply indebted to him for his constant guidance, valuable advice and patient reading of this thesis as well as other manuscripts.

I am grateful to my lovely brothers and sisters in Christ who support me in their prayers consistently, namely, Amos Chan, Tommy Chau, Winky Lai, Po Li, Anthony Ng, Cara Chan and Daisy Ngai. I also thank Jones Chui and Andrew Leung with whom I have had many enlightening discussions.

Finally, I would like to express my deepest gratitude to my family for their understanding and support throughout the past two years.

TABLE OF CONTENTS

LIST OF FIGURES vi

LIST OF TABLES viii

1 INTRODUCTION

1.1 Recent Resurgence of Artificial Neural Networks 1-1

1.2 A Design Problem in Applying Back-Propagation Networks 1-4

1.3 Related Works 1-6

1.4 Objective of the Research 1-8

1.5 Thesis Organization 1-9

2 MULTILAYER FEEDFORWARD NETWORKS (MFNs) AND BACK-PROPAGATION (BP) LEARNING ALGORITHM

2.1 Introduction 2-1

2.2 From Perceptrons to MFNs 2-2

2.3 From Delta Rule to BP Algorithm 2-6

2.4 A Variant of BP Algorithm 2-12

3 INTERPRETATIONS AND PROPERTIES OF BP NETWORKS

3.1 Introduction 3-1

3.2 A Pattern Classification View on BP Networks 3-2

3.2.1 Pattern Space Interpretation of BP Networks 3-2

3.2.2 Weight Space Interpretation of BP Networks 3-3

3.3 Local Minimum 3-5

3.4 Generalization 3-6

4 GROWTH OF BP NETWORKS

4.1 Introduction 4-1

4.2 Problem Formulation 4-1

4.3 Learning an Additional Pattern 4-2

4.4 A Progressive Training Algorithm 4-4

4.5 Experimental Results and Performance Analysis 4-7

4.6 Concluding Remarks 4-16

5 PRUNING OF BP NETWORKS

5.1 Introduction 5-1

5.2 Characteristics of Hidden Nodes in Oversized Networks 5-2

5.2.1 Observations from an Empirical Study 5-2

5.2.2 Four Categories of Excessive Nodes 5-3

5.2.3 Why are they excessive ? 5-6

5.3 Pruning of Excessive Nodes 5-9

5.4 Experimental Results and Performance Analysis 5-13

5.5 Concluding Remarks 5-19

6 DYNAMIC CONSTRUCTION OF BP NETWORKS

6.1 A Hybrid Approach 6-1

6.2 Experimental Results and Performance Analysis 6-2

6.3 Concluding Remarks 6-7

7 CONCLUSIONS 7-1

7.1 Contributions 7-1

7.2 Limitations and Suggestions for Further Research 7-2

REFERENCES R-1

APPENDIX

A.1 A Handwriting Numeral Recognition Experiment : Feature Extraction Technique and Sampling Process A-1

A.2 Determining the distance $d = \delta^2/2r$ in Lemma 1 A-2

LIST OF FIGURES

Figure 1.1	Nonlinear activation functions employed by ANN models	1-2
Figure 1.2	A three-layer feedforward network	1-4
Figure 2.1	The perceptron model	2-3
Figure 2.2	The perceptron in 2-D Euclidean space E^2	2-4
Figure 2.3	Linear partitions of four points in 2-D Euclidean space E^2	2-4
Figure 2.4	Sigmoid function f(x) with different smoothness factors : dotted line for t=0.25, solid line for t=1, and dashed line for t=4	2-6
Figure 2.5	Flowchart of the BP algorithm	2-11
Figure 3.1	(a) The Exclusive-OR problem separated by two solution nodes H1 & H2. (b) Hidden patterns formed by the two solutions are linearly separated by an output node L1.	3-3
Figure 3.2	A two-dimensional sample weight space with solution region R_s	3-5
Figure 3.3	Error surface in the weight space	3-6
Figure 4.1	Flowchart of the progressive training algorithm	4-7
Figure 4.2	Number of hidden nodes generated by the progressive training : N-bit parity problems	4-9
Figure 4.3	Generalization performance comparison of standard BP and progressive training : the IRIS data training sets	4-10
Figure 4.4	Number of hidden nodes generated by the progressive training : the IRIS data training sets	4-11
Figure 4.5	Learning speed comparison of standard BP and progressive training : the IRIS data training sets	4-11
Figure 4.6	Generalization performance comparison of standard BP and progressive training : the handwriting numeral training sets	4-13
Figure 4.7	Number of hidden nodes generated by the progressive training : the handwriting numeral training sets	4-13

Figure 4.8	Learning speed comparison of standard BP and progressive training : the handwriting numeral training sets	4-14
Figure 5.1	An 1-D classification example where patterns $x_1, x_2 \& x_5$ belong to class A and patterns $\overline{x}_3 \& \overline{x}_4$ belong to class B	5-3
Figure 5.2	(a) The weight space representation of the 1-D example consisting of ten regions (R1 to R5, R1' to R5'). (b) Decision hyperplanes in the input pattern space corresponding to each of the ten weight region with H3 & H5' denoting a set of solution nodes.	5-4
Figure 5.3	Hidden patterns formed by a solution node	5-7
Figure 5.4	Hidden patterns formed by two solution nodes that can be separated by an output node L1	5-7
Figure 5.5	Linearly non-separable hidden patterns formed by a solution and (a) a non-contributing node, (b) a duplicated node, (c) an inversely-duplicated node, and (d) an inadequate node	5-8
Figure 5.6	Linearly separable hidden patterns formed by two solution nodes and one inadequate node	5-9
Figure 6.1	Flowchart of the hybrid algorithm	6-2
Figure 6.2	Number of hidden nodes generated by the hybrid algorithm : *N*-bit parity problems	6-3
Figure 6.3	Number of hidden nodes generated by the hybrid algorithm : the IRIS data set	6-4
Figure 6.4	Number of hidden nodes generated by the hybrid algorithm : the handwriting numeral data set	6-5
Figure 6.5	Generalization performance of the hybrid algorithm : the IRIS data set	6-5
Figure 6.6	Generalization Performance of the hybrid algorithm : the handwriting numeral data set	6-6
Figure A.2.1	Schematic illustration of Lemma 1's proof	A-3

LIST OF TABLES

Table 4.1	Sensitivity of scale-up factor ζ (STEADYE=50)	4-15
Table 4.2	Sensitivity of STEADYE (ζ=10)	4-16
Table 5.1	Sensitivity of threshold parameters to pruning	5-13
Table 5.2	Node pruning simulation results of IRIS data set	5-15
Table 5.3	Node pruning simulation results of handwriting numeral data set	5-16
Table 5.4	Generalization performance before and after pruning	5-18
Table 5.5	Generalization performance before and after pruning (50-hidden-node networks)	5-18
Table 6.1	Network construction process using different sets of parameter values	6-7

1 INTRODUCTION

1.1 Recent Resurgence of Artificial Neural Networks

Today we can build digital computers that perform a variety of well-defined tasks with celerity and reliability unmatched by humans. No human can accomplish mathematical operations such as matrix inversions and solving systems of differential equations at speeds competitive with modern workstations. Nonetheless, many problems remain to be solved to our satisfaction by any man-made machine, but easily disentangled by the perceptual or cognitive powers of humans, or even fish and insects. No computer vision system, even a highly sophisticated one, can rival the human ability to analyse scenes characterized by objects of all shapes, orientations, and perspectives under a wide range of conditions. Humans effortlessly recognize objects in diverse environments and lighting conditions, even when obscured by dirt, or occluded by other objects. Our brains accomplish this by utilizing massive parallelism, with millions and even billions of neurons in parts of the brain working together to perform complicated tasks.

Thus, it is reasonable to consider solving certain problems by designing naturally parallel computers, which process information and learn by principles borrowed from the nervous systems of biological creatures. The point of convergence between biological nervous systems and artificial neural networks (ANNs) is that each typically consists of a large number of simple elements that learn and are able to collectively solve complicated and ambiguous problems. In fact, ANNs have been studied for many years in the hope of achieving human-like performance. It is until 1980s, with the success of new network models and learning algorithms [1-5], the field exploded. Experts from diverse disciplines such as physics, psychology, mathematics, engineering and computer science have been attracted to join the emerging field of ANNs. Therefore, rigourous and fascinating research works that lead to important discoveries and in-depth understanding of ANNs can be expected in the near future.

Artificial neural networks go by many names such as connectionist systems, neural

computers and parallel distributed processing systems. Whatever the name, all ANN models compose of simple neuron-like computational elements called nodes connected by links with variable weights. Each node has a "state" or "activation value" that is determined by the input received from nodes, including itself, in the network. One common, simplifying assumption is that the combined effects to a node j, which is called the "net input" of j, is usually taken to be a linear function of the states of the nodes that provide input to it, that is,

$$net_j = \sum_i w_{ji} x_i + \theta_j \tag{1.1}$$

where x_i is the state of node i, w_{ji} is the weights connected from node i to node j and θ_j, which can be considered as another weight from a node with fixed state at 1, is the threshold of node j. The state of node j is typically defined to be a nonlinear function of its net input net_j. Figure 1.1 illustrates three common types of nonlinearities employed by ANN models; hard limiters, threshold logic, and sigmoid function. Knowledge of ANNs is built up by where the connections are and by their weights, so learning takes place by modifying the weight values and/or changing their interconnections.

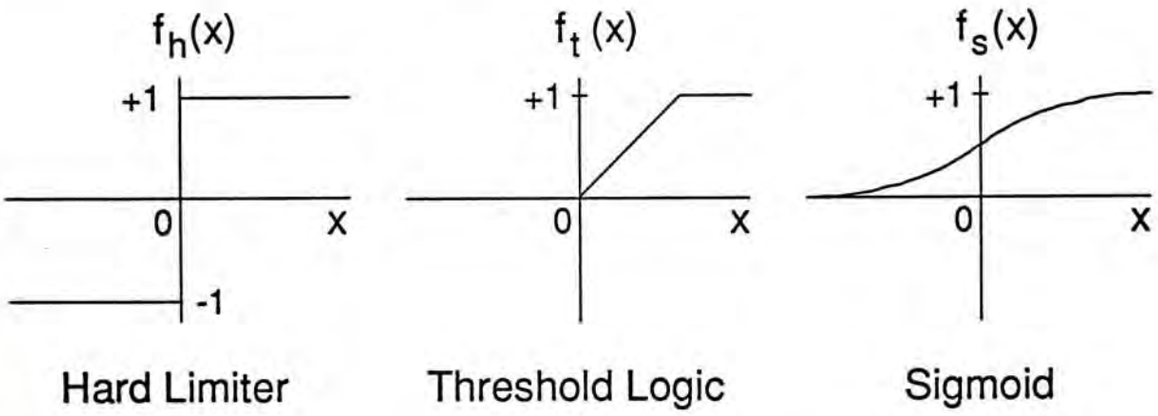

Figure 1.1 Nonlinear activation functions employed by ANN models

Neural network models are usually specified by their topologies and learning rules. Roughly speaking, they can be divided into three categories with each based on different philosophy. In *feedback networks* [1,6], the input pattern defines the initial state of the feedback system, and after state transitions the asymptotic final state is identified as the network's output. Among this category of network models, Hopfield's associative memory model [1]

should be the most well-known one. The second category is characterized by its *self-organizing* nature. Neighbouring nodes in the network compete in their activation values by means of mutual lateral interactions, and develop adaptively into specific detectors of different patterns. Important theories on this category were pioneered by Kohonen with his work on Self-Organizing Feature Map (SOFM) [3] and Grossberg with his development on Adaptive Resonance Theory (ART) [7-9]. The last category is the *feedforward networks* [4,10,11] which map sets of input patterns into sets of output patterns. The desired input-output mapping is usually determined by external, supervised adjustment of the network's weights. Without any doubt, the Back-Propagation (BP) networks, as formulated by Rumelhart et al. [4], may well be recognized as the most widely-used feedforward type model. In this thesis, we will concentrate on this type of ANNs.

The BP network is a hierarchical design consisting of fully interconnected layers of nodes. A three-layer network is shown in Figure 1.2. The layer receiving training patterns is called input layer while the one generating outputs for the network is called output layer. All layers in between are the hidden layers. The operation that BP networks are intended to carry out is approximating a bounded function $f: S \subset R^n \rightarrow R^m$, from a compact subset S of n-dimensional Euclidean space to a bounded subset $f(S)$ of m-dimensional Euclidean space, by means of training on examples $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k), \ldots$ of the mapping, where $y_k = f(x_k)$. As pointed out by Robert Hecht-Nielsen [30], the BP network is one of the most important historical developments in ANNs. Today, it is a mainstay of the field.

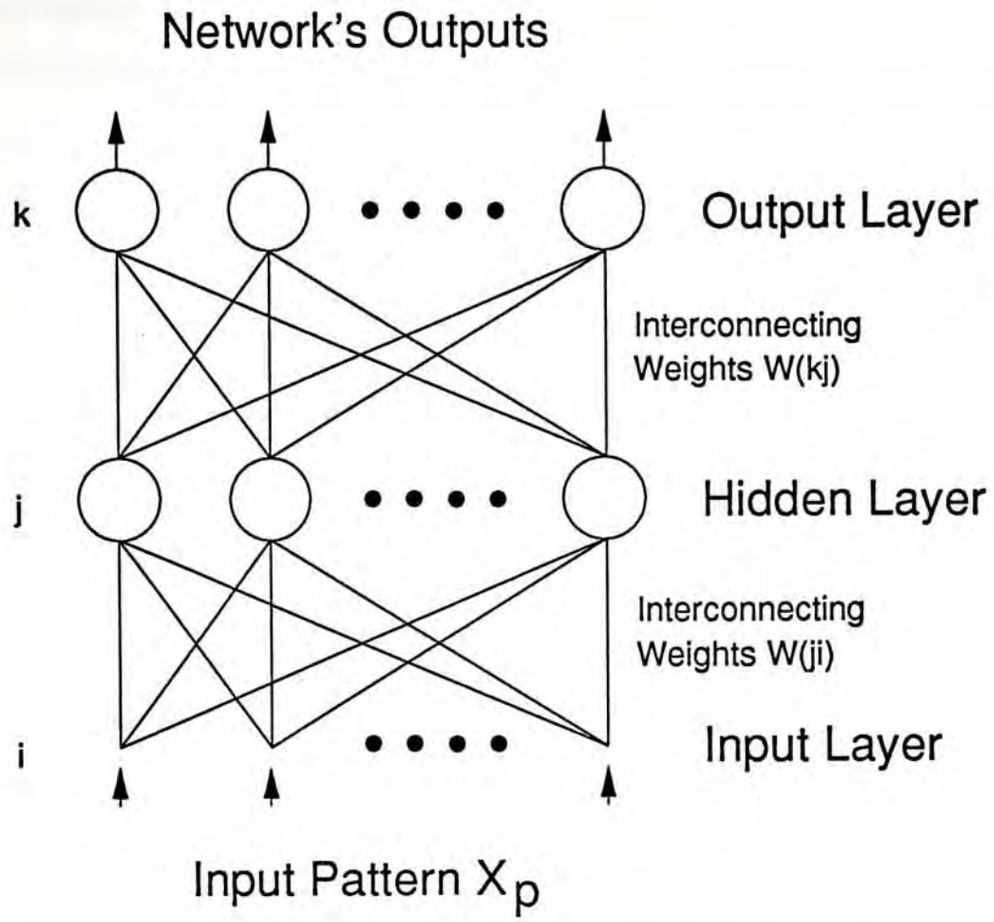

Figure 1.2 A three-layer feedforward network

1.2 A Design Problem in Applying Back-Propagation Networks

Since the promulgation of BP networks by Rumelhart et al. [5] in 1986, successful applications of the model to diverse areas such as phoneme recognition [12], machine vision [13], channel equalization [14] and system identifications [15] have been frequently reported. In the context of pattern classification for example, the performance of BP networks was found superior to that of the traditional statistical classifiers like Bayes, nearest-neighbor, and minimum-mean-distance [16,17,18]. The model not only offers better classifications on ordinary training data, but also works well with noisy data and in cases that only limited amount of training samples is available [17]. Despite of its success in various applications, the model itself is not free from problems. Three issues always bring researchers into discussions and they are (i) the learning speed, (ii) the problem of local minimum and, (iii) the structural requirement for a particular task.

Back-propagation learning was found too slow in some applications [19]. Furthermore, it scales poorly as the tasks grow larger and more complex [22]. Bundles of modified or new algorithms have been proposed to tackle this problem (see, for example, [19-22]), however, world-wide agreed optimal algorithm has not yet come. Another difficulty inherent in the BP learning is that it sometimes falls into a local minimum of the objective error function. However the problem is frequently ignored [29,53] and only a few investigations [28,43,53] have been appeared so far. Since the problem of local minimum is not the focus of this section, it will be recalled later on in Section 3.4.

A design problem that is often confronted in the application of BP networks is "how large is the network required to perform a particular task ?". Consequently, one may think of the following questions :

- How many layers should be used ?
- How to set the number of input and output nodes ?
- How many hidden nodes are needed ?

Recently, Hornik et al. [26] and Funahashi [27] have showed that BP networks with as few as one hidden layer which have sigmoid output functions can approximate virtually any function of interest, provided sufficiently many hidden nodes are available. This also implies that failures in applications can be attributed to inadequate training or insufficient number of hidden nodes rather than the capability of the network itself. In fact, many of the successful applications employed networks with one hidden layer only [16,17,25]. Therefore, three-layer networks with one for each of input, hidden and output nodes are good enough generally. Setting the number of input nodes however is quite a passive task because it depends on the feature representation of the problem, and hence the number of input nodes is usually set equal to the number of features. In some applications, determining the number of output nodes is trivial, for example, one output node is employed by time-series prediction [23], and for image compression, the number of output nodes must be equal to the number of input nodes [24]. The cases in pattern classification are somewhat different and two strategies are commonly adopted. One is that it is set equal to the number of pattern classes with each pattern class represented by one output node. Another strategy takes use of the binary

representation of pattern classes and only requires $\log_2 m$ output nodes for m pattern classes. As pointed out by Rajavelu et al. [25], the former strategy has an advantage of improved convergence but costs more network resources. Determining the appropriate number of hidden nodes perhaps is the major difficulty in applying BP networks. Practically speaking, one would rather overestimate than underestimate that number as an undersized network may never solve the problem (for example, the Exclusive-OR problem cannot be solved by a single node network). However, oversized networks have two major drawbacks : (i) it is well-known that, for a fixed amount of training data, networks with too many free parameters do not lead to good generalization, i.e., the ability to recognize patterns which have never been presented to the network before [31-33], (ii) the hardware realization is inefficient since more connections are required for larger networks. Therefore, attaining a BP network that is optimized with both the network size and performance is indeed essential to the applications of BP networks and is the central theme of this thesis.

1.3 Related Works

Recently, the network design problem stated in last section has received considerable attentions. One common approach to tackle the problem is to start with an oversized network and unnecessary nodes and links in the converged network are then removed afterwards. The difficulty of this *network pruning* approach is to identify those nodes and links to be removed while the network performance would not be significantly impaired due to the removals. Various methods have been proposed to accomplish this reduction, for example, several researchers [34,35] suggest to add an extra error term such as a function of the squared outputs of the hidden nodes [34] to the usual error function in order to cause non-essential links and nodes to decade away during the optimization process. The problem with this method, as pointed out and supported by the experiments in [35], is that more local minima would be introduced since two error landscapes are being added together and the solutions obtained may partially satisfy each of the error terms. On the other hand, Mozer and Smolensky [36] have introduced the idea of estimating the sensitivity of the error function to the elimination

of each node. Le Cun et al. [38] have independently developed a similar technique which operates on individual links instead and judiciously applies the second-derivative information of the objective error function to compute the sensitivity number for each link. The nodes or links with small sensitivity number, i.e. those contributing least to minimize the error function, are then pruned from the trained network iteratively until the pruning process would increase the error significantly. Unfortunately, precisely when to stop pruning is unclear and hence the performance of pruned network could be severely degraded without appropriate stopping criteria.

Another approach to handle the network design problem is by *network growth* [39,41,42]. In contrast to network pruning, this approach starts with a small network and allows it to grow by adding new nodes one by one in order to follow the peculiarities of training data. A major difficulty of this approach is how to add nodes effectively in such a manner that it can be guaranteed to converge. Interesting enough, many of the existing algorithms came from the works of theoretical physicists [39,41]. In [39], Mezard and Nadal propose a Tiling algorithm that adds nodes layer by layer onto the network's outputs. The most recently built layer gives a strictly better approximation of the desired outputs than the previous one, so eventually it gives the exact mapping. Nodes in the network are characterized by the hard-limiter non-linearity and each new node is generated through the Pocket algorithm, a variant of Perceptron algorithm [45]. The Upstart algorithm proposed by Frean [41] adopted a different way to construct the network. New nodes are interpolated between the input and output layer and each node takes on the unipolar hard-limiter activation function, producing either 0 or +1 output. The key idea of this algorithm is that a node can build other nodes to correct its mistakes. For example, if a node is "wrongly ON" (actual output is 1, but desired output is 0) for some patterns, it could be corrected by a large negative weight from a new node, which is ON only for those patterns. Likewise "wrongly OFF" (actual output is 0, but desired output is 1) mistakes could be corrected by using positive weight. Hence if the output nodes make errors, new hidden nodes are generated successively until the desired mapping is achieved. Again, a Perceptron-type algorithm was used to learn the interconnecting weights. Although these two algorithms are guaranteed to converge, they have

been restricted to consider generating networks to perform Boolean input-output mappings only and generalization to continuous mapping seems to be non-trivial. Fahlman and Lebiere [42], from the computer scientist point of view, have also developed a Cascade-Correlation learning algorithm which is quite similar to the Upstart algorithm except it works on networks composed of sigmoid nodes. Satisfactory performances are reported but no convergence proof is provided.

Recently, Hirose et al. [42] attacked the problem by changing the network size dynamically. The proposed algorithm simply adds nodes as the error no longer decreases. Once the network converges, a hidden node is removed, and the network is trained again. If the network converges again, another hidden node is removed. The procedure is repeated until the network no longer converges. Although this is quite a brute force technique, minimal networks are expected to construct without any guessing on the initial network size, even an oversized one. However, a big missing of this algorithm is that the growth process lacks of a convergence proof.

1.4 Objective of the Research

The objective of this work is to address the network design problem by devising an automatic network construction algorithm to attain an optimal network that is minimal in size and has good performance on both training and testing (unseen) data. Based on the arguments stated in Section 1.2, we had confined our study to networks with one hidden layer only and focused on determining the number of hidden nodes required to perform a particular task. Two different approaches were chosen to pursue in this research. By network growth, there is no need to guess the structure in advance and the required network is formed by a node adding process. If nodes are added without appropriate guideline, infinitely large networks could be resulted. Therefore, we aimed at deriving a guaranteed convergence algorithm that allows the network to grow from a small one and self-determine a reasonable structure for the available training data. However with network growth, the number of hidden nodes can only increase. In some cases, the number of hidden nodes might become

ridiculously large, so we had also opted for the network pruning approach and worked toward to derive an effective pruning algorithm that would eliminate unnecessary nodes and links while preserving the network performance such that an optimal or nearly optimal network could be obtained. By network pruning alone, *a prior* knowledge of the task such as its complexity is still required to predetermine a reasonably oversized network. Therefore, we sought the possibility to combine these two approaches in order to complement the individual weaknesses.

1.5 Thesis Organization

The thesis is made up of seven chapters. In this introductory chapter, a network design problem in applying BP networks has been posed, and the objective of the research has been stated. In the following chapter, the historical development of the architecture and learning algorithm of BP networks is reviewed. Chapter 3 introduces two interpretations of BP networks which will serve as a conceptual tool to derive a network pruning algorithm in Chapter 5. In addition, the local minimum problem and generalization property of BP networks are discussed. Chapter 4 is devoted to describe our network growth approach. Based on the characteristics of human learning and an observation of BP training, a progressive training algorithm to construct a reasonably large network is proposed. The performance of the proposed algorithm is reported and evaluated through various experiments. In Chapter 5, a study on the characteristics of hidden nodes in oversized networks, which leads to a new node pruning algorithm, is presented. Four categories of excessive nodes are identified from the results and an insight to why they are suitable candidates for pruning is also provided. The experimental results and performance analysis of the proposed node pruning algorithm are described at the end of the chapter. Chapter 6 advocates the idea to combine the proposed network growth and network pruning algorithms in order to dynamically construct the BP networks. The effectiveness of the proposed hybrid algorithm is demonstrated through vari-

ous experiments. The last chapter concludes the thesis by summarizing the contributions and limitations of the present work, and finally suggestion on potential areas for further research is given.

2 MULTILAYER FEEDFORWARD NETWORKS (MFNs) AND BACK-PROPAGATION (BP) LEARNING ALGORITHM

2.1 Introduction

Back-Propagation (BP) network is named after its learning rule --- BP algorithm [4]. In fact, BP has a strong historical background. Apparently, it was first developed by Paul Werbos [46] in 1974, as part of his Ph.D. thesis "Beyond regression : New tools for prediction and analysis in behavior sciences"; and independently rediscovered by David Parker [47] in 1985 and by David Rumelhart, Geoffrey Hinton and Ronald Williams [4] in 1986. A mathematically similar recursive control algorithm was presented by Arthur Bryson and Yu-Chi Ho [48] in 1969. Notwithstanding its checked history, there is no question that credit for developing BP into a usable technique, as well as broadcasting the algorithm to diverse audience, rests entirely with Rumelhart and his "PDP group" members [5]. Before their work, BP was unappreciated and obscure. On the topological side, BP network employs a multilayer feedforward structure which can be traced back to Nilsson's description of the layered machine [44] in 1965. Unfortunately, no practicable learning algorithm was available for such machines at that time. In this chapter, we will go through the historical development of such a fascinating artificial neural network (ANN) model in order to complete the presentation of the thesis. Those who are acquainted with the BP network may skip this chapter without any difficulty to understand the materials presented in the following chapters. Section 2.2 is devoted to describe the transcendency of the classical Perceptron model [45] to the more complex multilayer feedforward network (MFN). The delta rule for Perceptron learning and its extension to yield the BP algorithm are described in Section 2.3. The momentum strategy [4] which is a wide-spread variant of the BP algorithm to improve the learning speed is described in Section 2.4.

2.2 From Perceptrons to Multilayer Feedforward Networks

One of the most exciting developments during the early days of pattern recognition was the Perceptron [45], the idea that networks with layer(s) of Perceptron-like elements might be able to learn how to recognize and classify patterns in an autonomous manner. Correspondingly, one of the severe setbacks of early pattern recognition was the realization that simple single layer networks were inadequate for that purpose, and that multilayer ones lacked effective learning algorithms. Nevertheless, the Perceptron and the layered machine described by Nilsson [44] provided a solid conceptual base for further work in ANNs.

A Perceptron can be represented schematically in the form of an array of multipliers and summing junctions cascaded with a hard-limiter, as shown in Figure 2.1. In that illustration, the components of the input pattern $X_p = [x_{p1}, x_{p2}, \cdots, x_{pn}]$ are multiplied by a set of coefficients, the weight vector $W = [w_1, w_2, \cdots, w_n]$ and a unit input is multiplied by the bias weight θ. The sum of these weighted inputs is fed to the hard-limiter producing a binary ± 1 output. Mathematically, the Perceptron's output due to input pattern X_p is defined as

$$y_p = sgn\left(\sum_{i=1}^{n} x_{pi} w_i + \theta\right) \tag{2.1}$$

where $sgn(x)$ is the signum function. The components of X_p may be either continuous analog values or binary values. The weights are essentially continuously variable, and can take on negative as well as positive values.

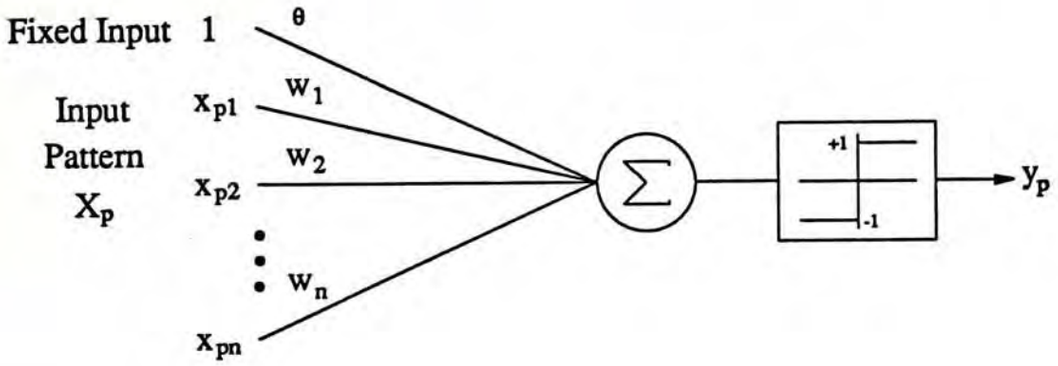

Figure 2.1 The perceptron model

A conventional technique for analyzing the behavior of Perceptrons is to plot the decision regions created in the multidimensional space spanned by the input patterns. These decision regions specify which input patterns result in a class A and which result in a NOT class A response. Due to its linear nature, the Perceptron forms two decision regions separated by a hyperplane. Figure 2.2 exemplifies how a Perceptron separates the 2-D input pattern space. The separating hyperplane (line) is defined by

$$x_2 w_2 + x_1 w_1 + w_0 = 0 \tag{2.2}$$

with the three weights w_0, w_1, and w_2 determining the slope, intercepts, and which side of the separating line corresponds to a +1 or -1 output. In this example, the Perceptron classifies pattern X_1 as one class with -1 output and patterns X_2, X_3, and X_4 as the other class having +1 response. There are $2^4 = 16$ different ways to categorize the four patterns in the 2-D space. However, only 14 of which can be implemented by the Perceptron, as shown in Figure 2.3 with each of the seven hyperplanes representing two opposite ways to classify the input pat-

terns. The other two cases are the well-known Exclusive-OR problems which have been used extensively (see, for example, [49,50]) to illustrate the inability of the Perceptron to solve nonlinear problems.

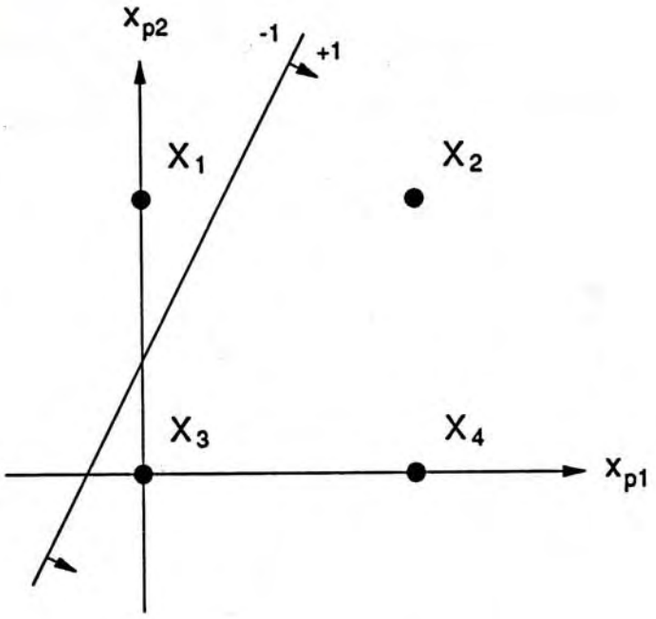

Figure 2.2 The perceptron in 2-D Euclidean space E^2

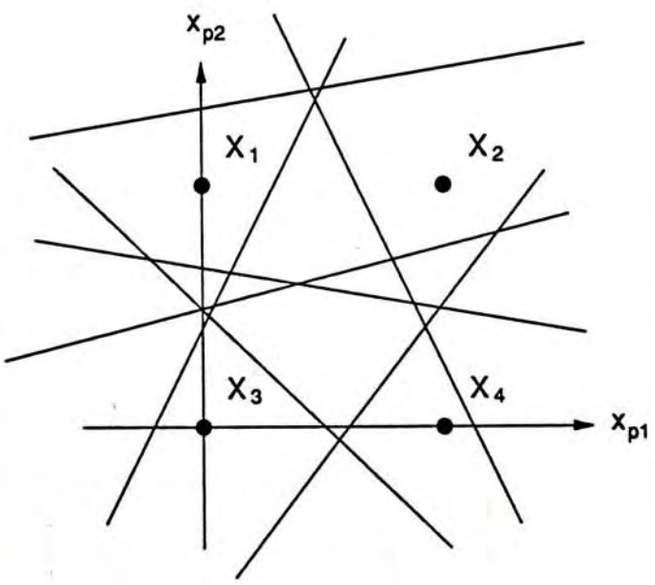

Figure 2.3 Linear partitions of four points in 2-D Euclidean space E^2

Owing to its severe limitations, the Perceptron was generalized to form the multilayer feedforward network which consists of sets of Perceptron-like elements called nodes arranged in layers. An example of this structure has already been shown in Figure 1.2. The network would have an input layer, an output layer, and any number of hidden layers in between. Let o_{pj} denote the output states of each node j due to pattern p. Then the output of node j is determined from output states of nodes i in the previous layer via the connecting weights w_{ji}, i.e.,

$$o_{pj} = f(net_{pj}) \tag{2.3}$$

where

$$net_{pj} = \sum_i w_{ji} o_{pi} + \theta_j \tag{2.4}$$

$$f(x) = \frac{1}{1 + \exp(-x/t)} \tag{2.5}$$

Like Perceptron, the bias term θ_j is usually treated as a normal weight component by assuming that it is a connection to an augmented node with output fixed at 1. The smoothness factor t determines how smooth the sigmoid function $f(x)$ is, and is usually set to one [4]. As t tends to zero, the sigmoid resembles the hard limiter, whereas a high value of t results in a more gently varying function. These are illustrated in Figure 2.4. The outputs of each layer are then fed to the input of higher layer and so on till the outputs are obtained. As a result, the functional form created by the MFN is a hierarchical sigmoided linear combinations of sigmoids which has been shown in [26,27] capable to approximate virtually any function of interest even the network under considerations consists of one hidden layer only. Although Nilsson [44] envisaged the capability of MFNs in solving hard problems more than twenty years ago, it is until the recent introduction of BP learning algorithm that such a powerful network can be trained effectively.

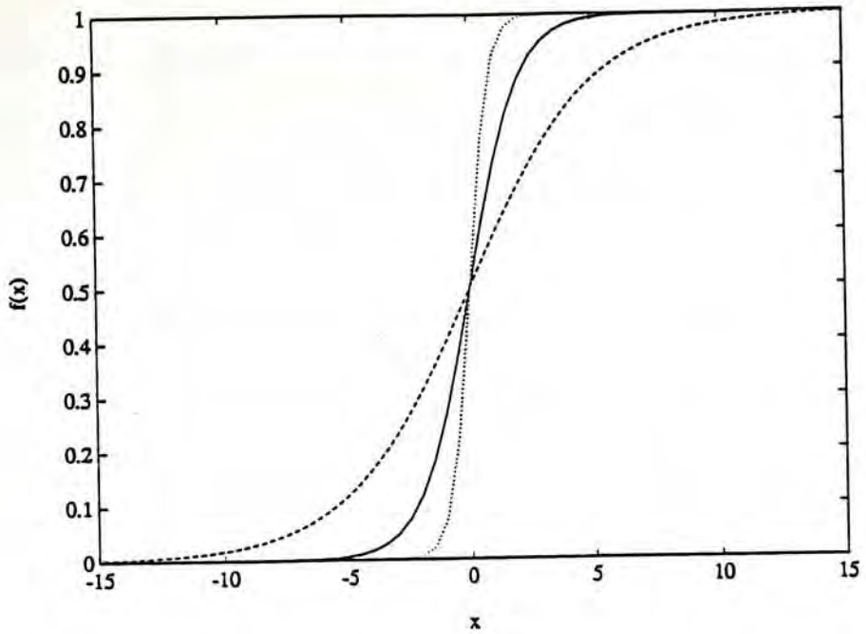

Figure 2.4 Sigmoid function f(x) with different smoothness factors : dotted line for t=0.25, solid line for t=1, and dashed line for t=4

2.3 From Delta Rule to Back-Propagation Algorithm

One of the predominant features of ANNs is "learning by examples". During the learning process of Perceptron, input patterns X_p and corresponding desired responses t_p are presented to the Perceptron. An adaptation algorithm automatically adjusts the weights so that the output responses y_p to the input patterns will be as close as possible to their respective desired responses. Such a Perceptron learning algorithm was first developed by Rosenblatt [45]. The connection weights are designed to adapt only when an error occurs. Specifically, the rule for changing weights following the presentation of input pattern X_p is given by

$$\Delta_p w_i = \eta(t_p - y_p)x_p \tag{2.6}$$

where η is a positive gain fraction less than one. Since t_p and y_p take on the value of [+1, -1], weights are unchanged if correct decision is made by the Perceptron. One problem with this learning rule is that the separating hyperplane may oscillate continuously when input patterns are not linear separable. The Widrow-Hoff delta rule or LMS algorithm [51], which is well-known to the signal processing community, modifies the Perceptron learning rule to a steepest descent type algorithm to find a least mean square (LMS) solution for the model.

The solution minimizes the half mean squared error between the desired response and the actual output, i.e.,

$$E = \frac{1}{2p} \sum_{p} (t_p - y_p)^2 \tag{2.7}$$

The derived weight adjustment rule is identical to equation (2.6) except the hard-limiter of the model is by-passed. Weights are thus corrected on every pattern presentation by an amount that depends on the difference between the desired response and the actual output.

Similar in spirit to the delta rule, the BP algorithm was designed to find a LMS solution for the MFN. Therefore, it is also termed as the generalized delta rule. Following the functional form of MFNs, the weight adjustment rules could be derived as follows and such a derivation is just a recapitulation of that in [4]. Let

$$E_p = \frac{1}{2} \sum_{k} (t_{pk} - o_{pk})^2 \tag{2.8}$$

and

$$E = \sum_{p} E_p \tag{2.9}$$

be the total half squared error between the desired response t_{pk} and the actual output o_{pk} of all the k output nodes due to the p-th pattern and due to all input patterns respectively. Applying the steepest descent process to minimize E_p, we have the interconnection weights being changed proportional to the derivative of the error measure with respect to each weight, i.e.,

$$\Delta_p w_{kj} = -\eta \frac{\partial E_p}{\partial w_{kj}} \tag{2.10}$$

where η determines the proportionality and is usually called the gain factor. Rewrite equations (2.3) & (2.4) for output node k as

$$o_{pk} = f(net_{pk}) \tag{2.11}$$

and

$$net_{pk} = \sum_{j} w_{kj} o_{pj} \tag{2.12}$$

respectively. Note that the bias term θ_j in expression (2.4) has been included as a normal weight in expression (2.12). The partial derivative $\partial E_p/\partial w_{kj}$ can be evaluated using the chain rule

$$\frac{\partial E_p}{\partial w_{kj}} = \frac{\partial E_p}{\partial net_{pk}} \frac{\partial net_{pk}}{\partial w_{kj}} \tag{2.13}$$

Using expression (2.12), we obtain

$$\frac{\partial net_{pk}}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}} \sum_j w_{kj} o_{pj} = o_{pj} \tag{2.14}$$

Now, let

$$\delta_{pk} = -\frac{\partial E_p}{\partial net_{pk}} \tag{2.15}$$

be a delta term of node k and expression (2.10) becomes

$$\Delta_p w_{kj} = \eta \delta_{pk} o_{pj} \tag{2.16}$$

which is similar in form to the delta rule of expression (2.6).

To compute $\delta_k = -\partial E_p/\partial net_{pk}$, we use the chain rule again, that is,

$$\delta_{pk} = -\frac{\partial E_p}{\partial net_{pk}} = -\frac{\partial E_p}{\partial o_{pk}} \frac{\partial o_{pk}}{\partial net_{pk}} \tag{2.17}$$

Using the expressions (2.8) & (2.11), we have

$$\frac{\partial E_p}{\partial o_{pk}} = -(t_{pk} - o_{pk}) \tag{2.18}$$

and

$$\frac{\partial o_{pk}}{\partial net_{pk}} = f'(net_{pk}) \tag{2.19}$$

Hence,

$$\delta_{pk} = (t_{pk} - o_{pk}) f'(net_{pk}) \tag{2.20}$$

for all output node k and hence

$$\Delta_p w_{kj} = \eta (t_{pk} - o_{pk}) f'(net_{pk}) o_{pj} \tag{2.21}$$

The situations in the hidden layers are somewhat different. Rewrite expressions (2.3) & (2.4) again for hidden node j as

$$o_{pj} = f(net_{pj}) \tag{2.22}$$

and

$$net_{pj} = \sum_i w_{ji} o_{pi} \tag{2.23}$$

respectively. We have

$$\begin{aligned}
\Delta_p w_{ji} &= -\eta \frac{\partial E_p}{\partial w_{ji}} \\
&= -\eta \frac{\partial E_p}{\partial net_{pj}} \frac{\partial net_{pj}}{\partial w_{ji}} \\
&= -\eta \frac{\partial E_p}{\partial net_{pj}} o_{pi} \\
&= \eta \delta_{pj} o_{pi} \\
&= \eta \left(-\frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial net_{pj}} \right) o_{pi} \\
&= \eta \left(-\frac{\partial E_p}{\partial o_{pj}} \right) f'(net_{pj}) o_{pi}
\end{aligned} \tag{2.24}$$

Since the outputs of hidden node j can contribute errors at all output nodes, the quantity

$$\begin{aligned}
-\frac{\partial E_p}{\partial o_{pj}} &= -\sum_k \frac{\partial E_p}{\partial net_{pk}} \frac{\partial net_{pk}}{\partial o_{pj}} \\
&= \sum_k \left(-\frac{\partial E_p}{\partial net_{pk}} \right) \frac{\partial}{\partial o_{pj}} \sum_j w_{kj} o_{pj} \\
&= \sum_k \delta_{pk} w_{kj}
\end{aligned} \tag{2.25}$$

Hence, the delta term of hidden node j becomes

$$\delta_{pj} = f'(net_{pj}) \sum_k \delta_{pk} w_{kj} \tag{2.26}$$

and the input-to-hidden interconnection weights are adjusted by the rule

$$\begin{aligned}
\Delta_p w_{ji} &= \eta \delta_{pj} o_{pi} \\
&= \eta \left[f'(net_{pj}) \sum_k \delta_{pk} w_{kj} \right] o_{pi}
\end{aligned} \tag{2.27}$$

Since the delta terms at each output node k can be evaluated using expression (2.20), we can then back-propagate these "errors" to the hidden layers and compute the expression (2.26) in a straight-forward manner.

Furthermore, if the sigmoid function of expression (2.5) takes place, then

$$f'(net_{pk}) = \frac{\partial o_{pk}}{\partial net_{pk}} = o_{pk}(1 - o_{pk}) \tag{2.28}$$

and the delta terms for the output and hidden nodes are computed as

$$\delta_{pk} = o_{pk}(1 - o_{pk})(t_{pk} - o_{pk}) \tag{2.29}$$

$$\delta_{pj} = o_{pj}(1 - o_{pj}) \sum_{k} \delta_{pk} w_{kj} \tag{2.30}$$

respectively. Expressions (2.21) & (2.27) form the core part of the BP algorithm for training MFNs. The network is trained initially selecting small random weights and then presenting all training data repeatly. Weights are adjusted after every trail until the convergence criteria is met. The flowchart for the BP algorithm is shown in Figure 2.5.

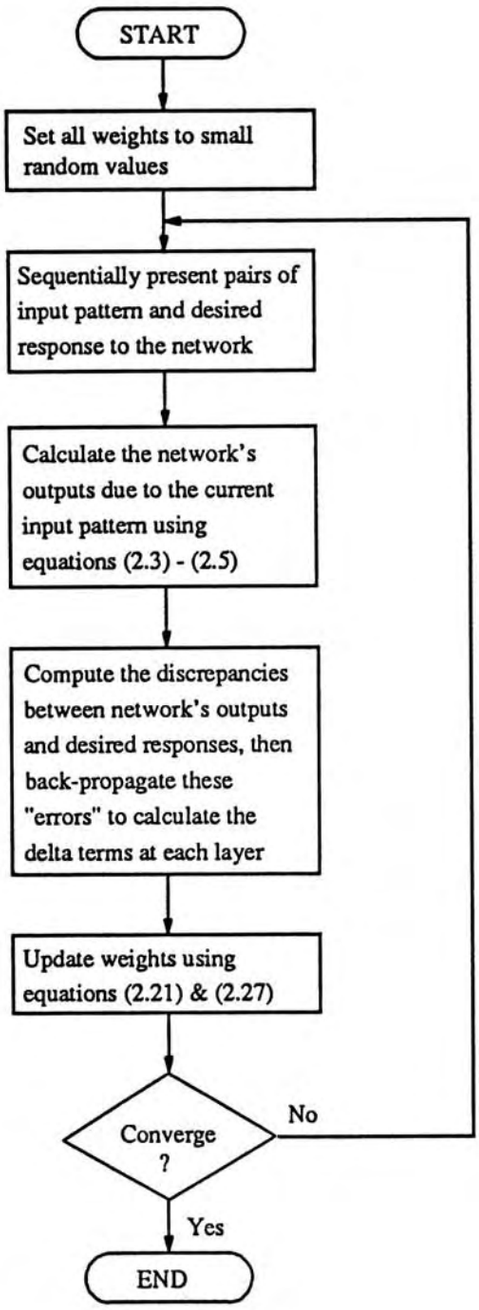

Figure 2.5 Flowchart of the BP algorithm

2.4 A Variant of BP Algorithm

A variation which is frequently used to increase the learning speed is to modify the weight adjustment rules to include a momentum term [4], i.e.,

$$\Delta_p w_{kj} = \eta(\delta_{pj} o_{pi}) + \alpha \Delta_{\hat{p}} w_{kj} \tag{2.31}$$

where $\Delta_{\hat{p}} w_{kj}$ is the change made to the weight w_{kj} following the presentation of previous input training pattern $X_{\hat{p}}$, α is the momentum factor that determines the relative contribution of the past weight change to the current weight change. This strategy has been used throughout the simulations of the research. The basic rationales behind the momentum strategy are that when the weight derivative possesses the same sign consecutively, the learning rate for that weight should be increased. When the sign of the derivative behaves in this manner, it is frequently the case that the error surface at the current point in weight space along that weight dimension possesses a small curvature, and therefore, continues to slope in the same direction for some significant distance. By increasing the learning rate for that weight, the time required for the value of this weight to traverse this distance can be reduced. On the other hand, when the sign of the derivative alternates for several consecutive time steps, the learning rate should be decreased. This phenomenon is frequently the case that the current error surface portion possesses a high curvature, and therefore, the slope of this area may quickly change sign. In order to prevent from oscillating, the weight should be adjusted by a smaller amount. As stated in [22], a speedup of a factor 2 to 3 might be expected by using the momentum strategy.

3 INTERPRETATIONS AND PROPERTIES OF BP NETWORKS

3.1 Introduction

One thing that ANNs have in common with human brains is that the internal organization is obscured by the immense number of nodes (neurons) and links (synapses). They work together in such a way that some desired global behavior is produced. However, one would like to have a better understanding of the role that individual nodes (and their corresponding links) play. Based on the classical pattern classification theory, Section 3.2 introduces two interpretations of BP network which will be used to derive a node pruning algorithm in Chapter 5.

Since BP is a gradient descent technique, the learning algorithm can get stuck in nearby local minimum. It has been observed that networks with hidden layers can have local minima in the error function when nonlinear functional nodes are used, causing the BP algorithm to fail [4], however this situation has been largely downplayed and ignored [29,53]. In Section 3.3, we discuss on this problem of BP networks and highlight existing approaches to solve the problem. Although most of the research works in BP networks have concentrated on improving the learning speed, an equally important feature of ANN is its generalization performance. It is usually accepted that good generalization performance on real-world problems cannot be achieved unless some *a prior* knowledge about the task is built into the system [31]. However, BP networks achieve this simply by learning from examples. In Section 3.4, we make a note on this property of BP networks and state a common belief that minimizing the number of nodes and links in the network enhances generalization. The local minimum problem and generalization property of BP networks will be referred from time to time in the remaining chapters of the thesis.

3.2 A Pattern Classification View on Back-Propagation Networks

In order to simplify the presentation of this section, a pattern classification problem with n-dimensional inputs and c classifications was assumed to be handled by a network having n input nodes and c output nodes; each corresponds to one classification. Each output node was considered as performing two-class classification by separating it from other classes and the target values of class k input patterns were assumed to be greater than 0.5 for output node k and less than 0.5 for other nodes.

3.2.1 Pattern Space Interpretation of BP networks

According to equation (2.4), net_{pj} is a linear discriminant function of pattern o_{pi} and it divides the pattern space of o_{pi} into two regions by a hyperplane defined as

$$\sum_i w_{ji} o_{pi} + \theta_j = 0 \tag{3.1}$$

It will be referred as a decision hyperplane. Consider a set of input patterns represented by n-dimensional Euclidean vectors in R^n. A node in the hidden layer can then be considered as a (n-1)-dimensional decision hyperplane that divides the input pattern space into two regions with one corresponding to output states >0.5 and the other <0.5. Also, the output state of each node in the hidden layer can be regarded as the components of a vector. If there are h hidden nodes, the hidden layer non-linearly transforms each n-dimensional input pattern vector into a h-dimensional vector with component values between 0 & 1. For convenience of distinction, those h-dimensional vectors will be referred as hidden patterns. Similarly, the output node divides the hidden pattern space into two regions by a (h-1)-dimensional decision hyperplane. In order to produce the desired output responses for each input patterns, class k hidden patterns should be linearly separated from other classes by output node k for all k. *Therefore, training of a BP network can be viewed as a problem of adjusting the decision hyperplanes of hidden nodes such that the transformation implemented by the hidden layer results in linearly separable hidden patterns at the output layer.* Such an interpretation of BP network can be exemplified by the Exclusive-OR problem as follows. Suppose a BP network

is trained to have hidden nodes' decision hyperplanes like those (H1 & H2) in Figure 3.1(a). The arrow of each hyperplane is pointing toward the region with output states >0.5 while the other side corresponds to <0.5. If the smoothness factor, t is sufficiently small, the distribution of hidden patterns will be similar to those depicted in Figure 3.1(b). Thus, they can be linearly separated by an output node L1 which gives desired output states for all the four patterns. A similar interpretation of BP networks can also be found in [50]. Brady et al. [29] has also adopted this interpretation to study the local minimum property of BP networks.

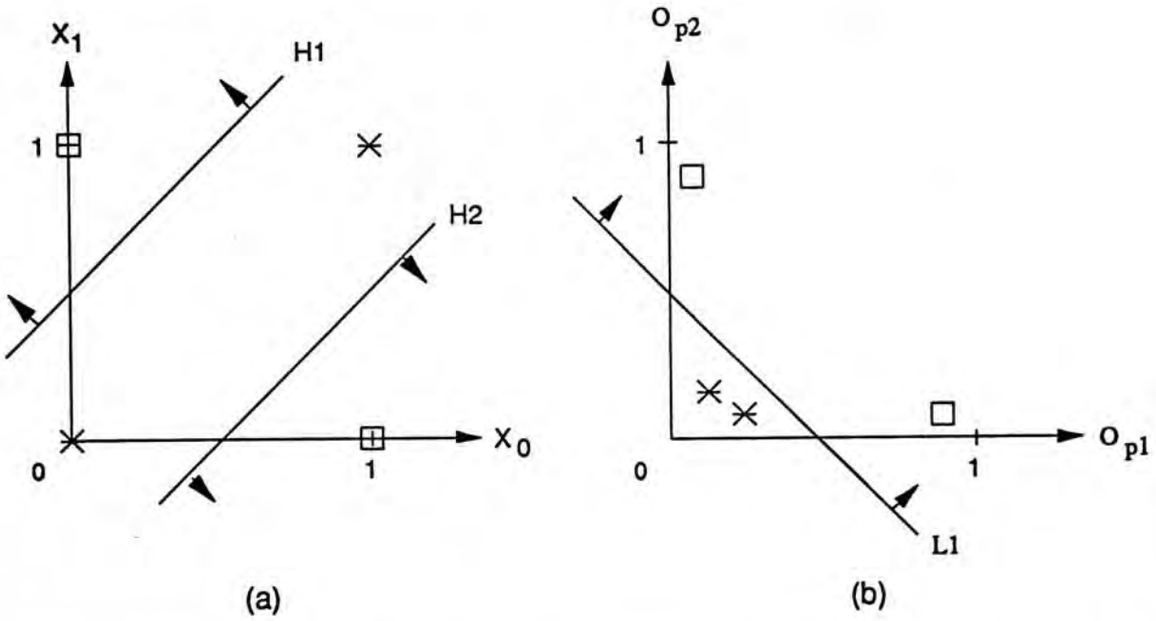

Figure 3.1 **(a) The Exclusive-OR problem separated by two solution nodes H1 & H2. (b) Hidden patterns formed by the two solution nodes are linearly separated by an output node L1.**

3.2.2 Weight Space Interpretation of BP networks

The dynamics of BP networks may also be studied within the domain of weight space [52,54] which is spanned by the weight components of each node in the network. Suppose now that there is a set of patterns $\{\vec{x}_1,\vec{x}_2,\ldots,\vec{x}_{p_0}\}$ in R^n from class A & B and a single node is used to classify them. A pattern $\vec{x}_p$ is classified correctly if $o_{pj} > 0.5$ (i.e. $net_{pj} > 0$) when $\vec{x}_p$

belongs to class A, and $o_{pj} < 0.5$ (i.e. $net_{pj} < 0$) when $\vec{x}_p$ is class B pattern. From equation (2.4)

$$net_{pj} = \vec{w}_j \cdot \vec{o}_p \tag{3.2}$$

where $\vec{w}_j = [w_{j1} \ldots w_{jn} \ \theta_j]$ & $\vec{o}_p = [x_{p1} \ldots x_{pn} \ 1]$. Therefore each pattern $\vec{o}_p$ defines a hyperplane in the $(n+1)$-dimensional weight space on which

$$\vec{w}_j \cdot \vec{o}_p = 0 \tag{3.3}$$

This hyperplane will be referred as a pattern hyperplane to distinguish from the decision hyperplanes defined earlier. The pattern hyperplane would pass through the origin and divides the weight space into two half-spaces. The half-space having weight vectors correctly classify $\vec{x}_p$ is called the proper region. For example, if $\vec{x}_p$ is from class A, the proper region of its pattern hyperplane would consist of all weight vectors satisfying the condition $\vec{w}_j \cdot \vec{o}_p > 0$. The set of p_0 input patterns thus imposes p_0 constraints on the possible location of a solution weight vector. The solution vector, if exists, must satisfy all the constraints is therefore located in the intersection of proper regions of all hyperplanes. The intersection region thus defines the solution region. If such a region exists, the patterns are said to be linearly separable and only require a single node to classify them. A two-dimensional example in weight space representation is depicted in Figure 3.2. The proper region of each pattern hyperplane is defined by the arrow on the hyperplane, thus the shaded region R_S is the solution region. Other regions correspond to different dichotomies of the patterns. Therefore, such a weight space representation can be used to describe the characteristics of a node in classifying input patterns. Note that the same representation can also be used to explore the relationship between the output node and hidden patterns. However, it is convenient to study the relationship between the hidden layer and the input layer through weight space representation while the relationship between the output layer and the hidden layer is explored in the pattern space domain. This is the strategy adopted in Chapter 5 to study the characteristics of hidden nodes in oversized networks.

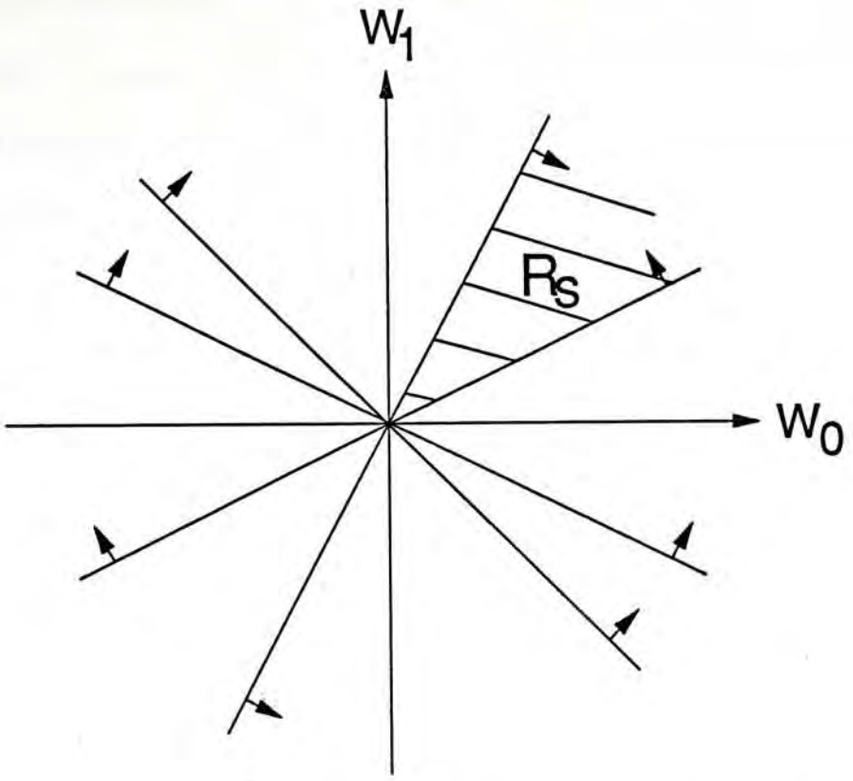

Figure 3.2 A two-dimensional sample weight space with solution region R_s

3.3 Local Minimum

The problem of local minimum is one of the three critical issues of BP networks mentioned in the introductory chapter. It concerns with the question of whether the network may get trapped in some local minimum or even at some stationary point, or perhaps oscillate between such points. Under such circumstances, the network's total error E remains large regardless of how many training iterations are carried out. This situation is depicted schematically in Figure 3.3. In that figure, the total error E is plotted against a weight component w. The objective of BP training is to find the weight w_{global} but the steepest descent process might get stuck at point w_{local} or even at point $w_{stationary}$. A common approach to deal with the problem is to add noise to the optimization process which provides a means of escaping a local minimum when encountered [2]. This approach requires a long convergence time, since many of the parameter (weight) adjustments do not decrease but rather increase the error in order to jump out of the potential local minimum regions. Baba [28] adopted this approach

by suggesting to use the random optimization method to train the network and successful results were reported. Another approach is to detect the occurrence of local minimum cases, and to escape by adding more nodes to the network so as to enlarge the dimensionality of the parameter (weight) space and hence provide more paths for the optimization process to converge to the global minimum [43,53]. In fact, the network growth algorithm proposed in the following chapter took use of the node adding mechanism on one hand to create the required network structure and on the other hand to get rid of the local minimum problem.

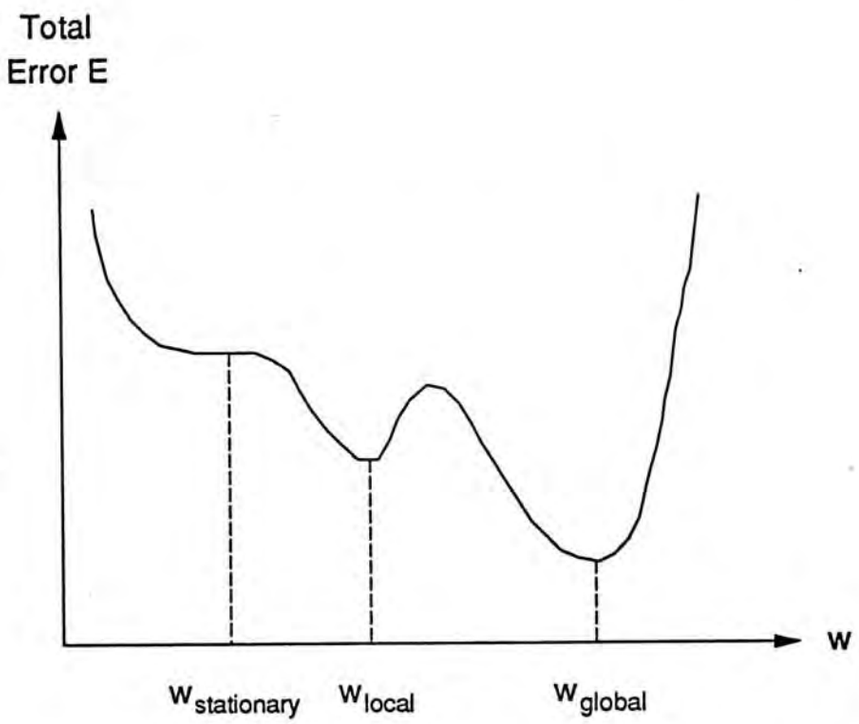

Figure 3.3 Error surface in the weight space

3.4 Generalization

Generalization is one of the capabilities expected for ANNs. It concerns with the ability to produce correct response when presented with patterns outside the training set. Theoretical study has shown that the likelihood of correct generalization depends on the size of the hypothesis space (total number of networks being considered), the size of the solution space (set of networks that give good generalization), and the number of training examples [33]. If

the hypothesis space is too large and/or the number of training examples is too small, then there will be a vast number of networks which are consistent with the training data, only a small proportion of which will lie in the true solution space, so poor generalization is to be expected. Conversely, if good generalization is required, when the dimensionality of the hypothesis space is increased, the number of training data must also be increased.

An illuminating analogy can be drawn between BP learning and curve fitting. When using a curve model (say a polynomial) with lots of parameters compared to the number of points, the fitted curve will closely model the training data but will not be likely to accurately represent new data. On the other hand, if the number of parameters in the model is small, the model will not necessarily represent the training data but will be more likely to capture the regularity of the data and extrapolate (or interpolate) correctly. When the data is not too noisy, the optimal choice is the minimum size model that represents the data.

From this analogy one may expect that minimizing the number of free parameters in the network increases the likelihood of correct generalization. But this must be done without reducing the size of the network to the point where it can no longer represent the desired function.

4 GROWTH OF BP NETWORKS

4.1 Introduction

In this chapter, the network growth approach to determine the number of hidden nodes required to perform a particular task is pursued. In our network growth strategy, hidden nodes are added one by one whenever they are needed during a well-ordered training process. The proposed algorithm called *progressive training* starts training with a one-hidden-node network and an initial training subset consisting of two input patterns only. The training subset is then expanded by including one more pattern and the previously trained network is trained again to cater for the new pattern. Such a process continues by progressively adding more and more patterns to the training subset until all the available training patterns have been taken into accounts. At each training stage, convergence is guaranteed and at most one hidden node would be added to the previously trained network. Thus the proposed algorithm can always find a solution network with finite number of hidden nodes for that particular task.

This chapter is organized into six sections. The problem is firstly formulated in the next section. Section 4.3 describes the basic rationale of the proposed algorithm and followed by introducing the concept of learning an additional pattern by a previously trained network. It proves that there exists an expanded-by-one solution network for both the additional pattern and the original patterns. The concept is generalized in Section 4.4 and from which the progressive training algorithm and its convergence proof are derived. Section 4.5 reports on the experimental results and the performance of the proposed algorithm is evaluated. Concluding remarks are made in the final section.

4.2 Problem Formulation

We now formulate the problem more precisely. Suppose we are given a training data set $S = \{\vec{x}_1, \vec{x}_2, \cdots, \vec{x}_{p_0}\}$ in E^n and we want to find a h-hidden-node BP network producing the set of desired outputs $S' = \{\vec{t}_1, \vec{t}_2, \cdots, \vec{t}_{p_0}\}$ for each of the training patterns in S. The network

under considerations here only has one hidden layer. Each node is characterized by the interconnecting weights and threshold leading to it. Let hidden nodes $j(=1,...,h)$ and output nodes $k(=1,...,m)$ have weight vectors $\vec{w}_j = \{\theta_j, w_{j1}, w_{j2}, \cdots, w_{jn})$ and $\vec{w}'_k = \{\theta'_k, w'_{k1}, w'_{k2}, \cdots, w'_{kh})$ respectively. For each of the input training patterns $\vec{x}_p$, their outputs are obtained by the rules

$$o_{pj} = f\left(\sum_{i=1}^{n} w_{ji} x_{pi} + \theta_j\right) \tag{4.1}$$

and

$$o'_{pk} = f\left(\sum_{j=1}^{h} w'_{kj} o_{pj} + \theta'_k\right) \tag{4.2}$$

with the activation function f defined earlier in expression (2.5) as

$$f(x) = \frac{1}{1+\exp(-x/t)} \tag{4.3}$$

Recall that function f will take on the characteristics of a hard limiter if t tends to zero. The task is accomplished by determining a proper value of h and the corresponding weight vectors $\vec{w}_j$ and $\vec{w}'_k$ such that $\vec{o}'_p$ complies with the desired output vector $\vec{t}_p$ for all p.

Before proceeding to the following sections, let us formally define the term solution network which has already been used previously and will be employed frequently later on.

Definition 1 : Let $S = \{\vec{x}_1, \vec{x}_2, \cdots, \vec{x}_{p_0}\}$ be a p_0-pattern training data set in E^n. A *solution network* of S is defined as the trained network whose output vectors $\vec{o}'_p$ are conformed with the desired output vectors $\vec{t}_p$ for each of the training patterns $\vec{x}_p$ in S.

4.3 Learning an Additional Pattern

A common characteristic of human learning is that it is easier to learn with smaller set of examples first and based upon which knowledge is accumulated gradually. Hence it is also reasonable to train the ANNs firstly with a small portion of the available training data and then progressively learn more and more using the previously trained networks. This strategy not only eases of learning but also provides a mechanism for the network to grow as the complexity of the training patterns increases. In this section, we concentrate on how a

trained network is modified in order to learn an additional pattern. In fact, the BP algorithm has been observed to converge quickly when it is applied to update a network to account for an additional training pattern [4]. Two outcomes of the updating exercise are possible. The BP algorithm converges successfully in which case the updating exercise is accomplished. Conversely, we can add a sub-network to handle the added pattern while keeping the original network to handle the old data. The simplest sub-network would be a single node in the hidden layer and we will show that it is capable to compensate for the introduced error if the added pattern is chosen in a special way.

Let us now give the proof. Without loss of generalization, the lemma and theorem given in this chapter assume that the activation function of each node is characterized by the hard limiter for presentation simplicity.

Lemma 1 : Given a new pattern $\vec{x}_{l+1}$ and a h-hidden-node solution network of a l-pattern training subset S_l, there exists a (h+1)-hidden-node solution network for the (l+1)-pattern training subset $S_{l+1} = S_l \cup \{\vec{x}_{l+1}\}$ if $||\vec{x}_{l+1}|| > ||\vec{x}_i|| \quad \forall \vec{x}_i \in S_l$. Furthermore, one can construct explicitly one such network by simply assigning deterministic values to the connection weights and thresholds.

Proof : Let $||\vec{x}_{l+1}|| = r$ and $\min_{\vec{x}_i, \vec{x}_j \in S_{l+1}} ||\vec{x}_i - \vec{x}_j|| = \delta$. Construct a closed hypersphere $\overline{C}(o,r)$ in E^n with radius r and center at the origin o. Since $||\vec{x}_{l+1}|| > ||\vec{x}_i|| \quad \forall \vec{x}_i \in S_l$, S_l is contained by $\overline{C}(o,r)$ and $\vec{x}_{l+1}$ lies on the surface of $\overline{C}(o,r)$. Construct another closed hypersphere $\overline{C}(\vec{x}_{l+1}, \delta)$ in E^n with radius δ and center at $\vec{x}_{l+1}$ and let I in E^n be the set of elements that characterizes the surface intersection of $\overline{C}(o,r)$ and $\overline{C}(\vec{x}_{l+1}, \delta)$. Consider now a tangent hyperplane $H = \{x: \vec{x}_{l+1} \cdot \vec{x} - \vec{x}_{l+1}^2 = 0\}$ in E^n to $\overline{C}(o,r)$ at $\vec{x}_{l+1}$. The distance between H and any element in I will be $d = \frac{\delta^2}{2r}$ (see Appendix A.2 for mathematical derivation). If H is translated onto the origin with distance less than d to $H' = \{x: \vec{x}_{l+1} \cdot \vec{x} - \vec{x}_{l+1}^2 \left(1 - \frac{\alpha d}{r}\right) = 0$ where $0 < \alpha < 1\}$ which partitions E^n into two open halfspaces, i.e., $H_+' = \{x: \vec{x}_{l+1} \cdot \vec{x} - \vec{x}_{l+1}^2 \left(1 - \frac{\alpha d}{r}\right) > 0\}$ and $H_-' = \{x: \vec{x}_{l+1} \cdot \vec{x} - \vec{x}_{l+1}^2 \left(1 - \frac{\alpha d}{r}\right) < 0\}$ where $0<\alpha<1$, all patterns of S_l will be in H_-' while the new pattern $\vec{x}_{l+1}$ will be in H_+'. Therefore, a new hidden node,

with weight vector $\vec{w}_{h+1} = \vec{x}_{l+1}$ and threshold $\theta_{h+1} = -\vec{x}_{l+1}^2\left(1 - \frac{\alpha d}{r}\right)$ where $0<\alpha<1$, can be added to the original h-hidden-node network such that it will not respond, i.e. producing zero output, to those patterns in S_{l+1} except the new pattern $\vec{x}_{l+1}$. As a result, this $(h+1)$-hidden-node network maintains the desired outputs for the original training subset S_l no matter what values the connection weights between the new node and output layer $w'_{k(h+1)}$ are. Furthermore, desired outputs for the new pattern $\vec{x}_{l+1}$ can be obtained from this enlarged network by setting

$$w'_{k(h+1)} = f^{-1}(t_{(l+1)k}) - \sum_{j=1}^{h} w'_{kj} o_{(l+1)j} + \theta'_k \tag{4.4}$$

for all $k=1,...,m$ where f^{-1} represents the inverse of f.

Q.E.D.

Note that this proof holds true for any desired one-to-one or many-to-one continuous mapping $\vec{x}_p \rightarrow \vec{t}_p$ where $\vec{x}_p \in R^n$ and $\vec{t}_p \in R^m$.

4.4 A Progressive Training Algorithm

Lemma 1 has provided a solid conceptual basis for the development of a new network growth algorithm --- progressive training. By generalizing Lemma 1, a one-hidden-node network is initially used to learn a (l=2)-pattern training subset. Upon convergence, the solution network is used to learn one more pattern and the updating strategy described earlier is applied. According to Lemma 1, at most one hidden node would be added to the original network if the magnitude of the added pattern is larger than that of previously learned ones. By conforming with the magnitude requirement, the updating exercise can be repeatly performed so as to learn all the available training patterns. In other words, the progressive training algorithm takes on a fixed training sequence in which patterns are ordered in ascending magnitude and the first two patterns are chosen as the initial training subset. A convergence proof of the proposed algorithm is given below.

Theorem 1 : Let $S = \{\vec{x}_1, \vec{x}_2, \cdots, \vec{x}_{p_0}\}$ be a p_0-pattern training data set in E^n. A (p_0-1)-hidden-node network is capable to produce the desired outputs for each of the training patterns $\vec{x}_p$ in S.

Proof : Consider that the elements of S are sorted in ascending magnitude order, i.e., $||\vec{x}_1|| \leq ||\vec{x}_2|| \leq \cdots \leq ||\vec{x}_{p_0}||$. Let $S_2 = \{\vec{x}_1, \vec{x}_2\}$. A one-hidden-node solution network of S_2 can be constructed as follows : Let h denote the single hidden node. Set $\vec{w}_h$ and θ_h such that h produces a zero output for $\vec{x}_1$ and a unit output for $\vec{x}_2$. Obviously, any $\vec{w}_h$ and θ_h implementing a separating hyperplane $H = \{x: \vec{w}_h \cdot \vec{x} + \theta_h = 0\}$ of $\vec{x}_1$ and $\vec{x}_2$ such that $\vec{x}_2$ is contained in $H_+ = \{x: \vec{w}_h \cdot \vec{x} + \theta_h > 0\}$ and $\vec{x}_1$ is contained in $H_- = \{x: \vec{w}_h \cdot \vec{x} + \theta_h < 0\}$ can achieve this objective. Now, for each of $\vec{x}_1$ and $\vec{x}_2$, the corresponding output at each output node k would be

$$o'_{1k} = f(\theta'_k) \tag{4.5}$$

and

$$o'_{2k} = f(w'_{kh} + \theta'_k) \tag{4.6}$$

In order to produce the desired outputs for $\vec{x}_1$ and $\vec{x}_2$, the values of connection weights w'_{kh} and thresholds θ'_k are selected as

$$\theta'_k = f^{-1}(t_{1k}) \tag{4.7}$$

and

$$w'_{kh} = f^{-1}(t_{2k}) - f^{-1}(t_{1k}) \tag{4.8}$$

Hence, a one-hidden-node solution network has been built for the two-pattern training data set S_2. By Lemma 1, an expanded-by-one solution network can always be found for each new pattern chosen in order from $\{\vec{x}_3, \vec{x}_4, \cdots, \vec{x}_{p_0}\}$ which is a (p_0-2)-pattern data set. As a result, a (p_0-1)-hidden-node solution network of S will be obtained.

Q.E.D.

Theorem 1 provides the theoretical upper bound on the number of hidden nodes constructed by the proposed algorithm. Clearly one would like to generate a network with minimal architecture. Having this in mind, we must consider when a new hidden node should be added. A reasonable approach to achieve this is to monitor the total error E as defined in expression (2.9). If it does not decrease by a reasonable amount after a prescribed number of training cycles, a new node is added. This strategy was adopted by the progressive training algorithm. In addition, we have to consider how a new hidden node is added. In fact, Lemma 1 has offered a skeleton to do so and the major concern here is to replace the hard limiter assumed in Lemma 1 by the usual sigmoid function taken by BP networks. Following the idea of the proof, if the total error E no longer decreases, the previously trained network is restored to handle the old data and a new hidden node h+1 is added and initialized to marginally separate the added pattern $\vec{x}_{l+1}$ from the old patterns, that is

$$\vec{w}_{h+1} = \vec{x}_{l+1} \tag{4.9}$$

$$\theta_{h+1} = -\vec{x}_{l+1}^2, \tag{4.10}$$

and the weights connecting the new hidden node to the output layer are simply initialized by equation (4.4). Thus hidden node h+1 will produce a 0.5 output for pattern $\vec{x}_{l+1}$ and some small positive values (<0.5) for the other patterns. This requires the BP algorithm to further adjust the network's weight so as to find a solution network. However, BP is a steepest descent type algorithm, it does not guarantee to find a global minimum solution. Therefore, a further step is designed to ensure convergence. It scales up the magnitudes of $\vec{w}_{h+1}$ and θ_{h+1} if the previous weight initialization does not lead to find a solution network, i.e.,

$$\vec{w}_{h+1} = \zeta^r \vec{x}_{l+1} \tag{4.11}$$

and

$$\theta_{h+1} = -\zeta^r \vec{x}_{l+1}^2 \tag{4.12}$$

where ζ specifies the scale-up factor and r is the frequency of scale-up request. Thus the sigmoid function of hidden node h+1 may eventually act like a hard limiter which in turn will produce negligible outputs for the trained patterns and hence convergence is guaranteed. The

flowchart of the progressive training is shown in Figure 4.1.

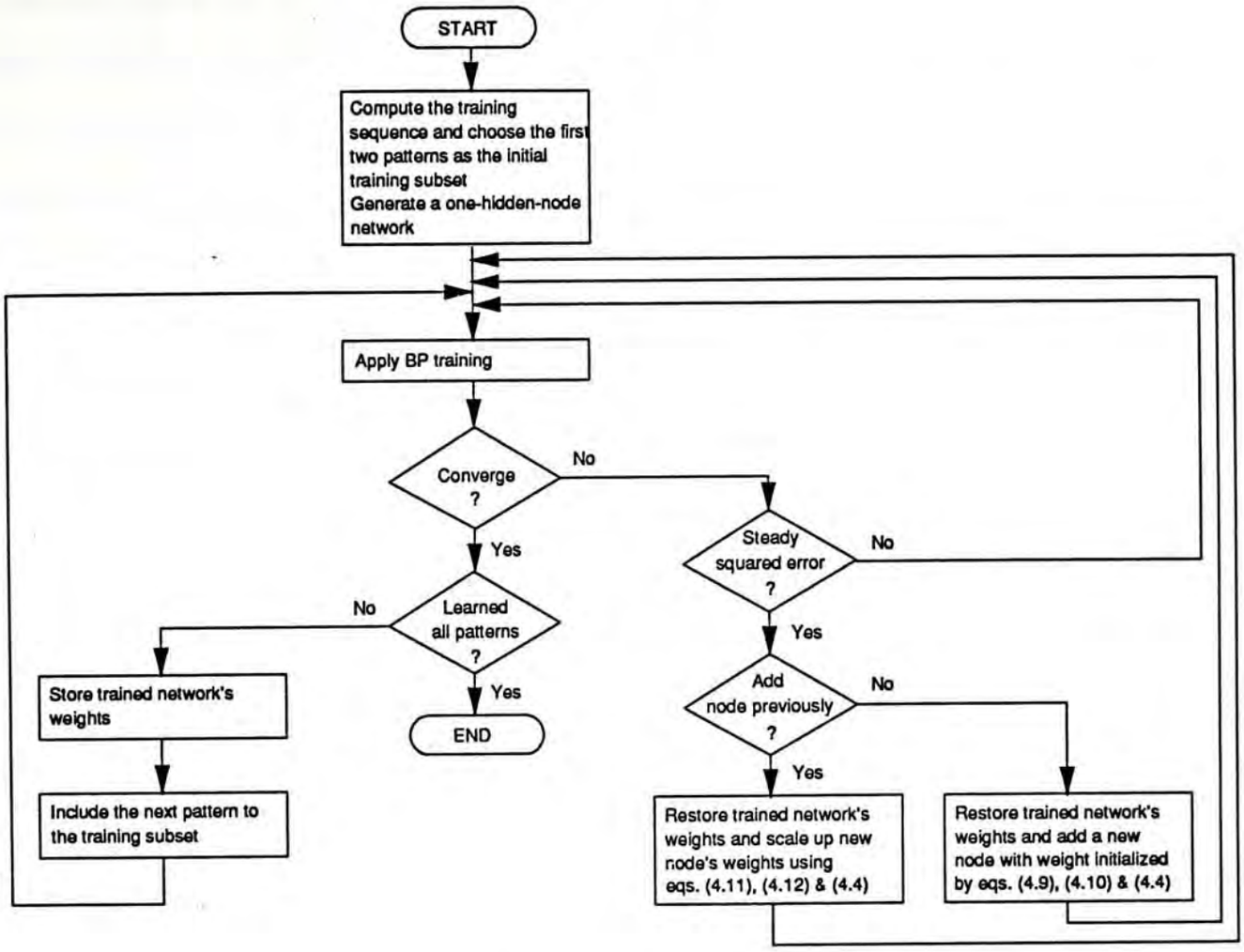

Figure 4.1 Flowchart of the progressive training algorithm

4.5 Experimental Results and Performance Analysis

Three experiments were carried out to study the performance of the progressive training algorithm. The one-hidden-layer networks were employed in all simulations and the numbers of input and output nodes were set equal to the numbers of features and pattern classes respectively. The magnitudes of training patterns were normalized to the range [-1,+1] and the initial one-hidden-node network's weights were randomly set between -0.5 & +0.5. At each training stage, the current training subset was considered learned when all the network's outputs were conformed with the target values ≥ 0.5& < 0.5 instead of the usual values 0.9 & 0.1 in order to minimize the training time. A new hidden node is added or its weights are

scaled up if the total error E does not decrease by more than one percent after a fixed number of *epoches* (an *epoch* being one traversal of the training set). The sensitivity of this parameter, named as STEADYE, and the scale-up factor ζ to the performance of progressive training will also be analysed in this section. For comparison purposes, standard BP trainings with fixed topologies were performed with all the interconnecting weights initialized between -0.5 & +0.5. Trainings were terminated when a solution was found or the maximum number of epoches was reached. All the other conditions are the same as that of progressive training. Unless otherwise stated, the gain and momentum factors were set to 0.5 and 0.7 respectively in all simulations.

(i) N-bit Parity

The parity problems have been popular benchmarks in the neural network community. It is often cited as a difficult problem for neural networks to learn because by changing one bit of information the output would be totally different. It is also of interest because one hidden layer networks with N hidden nodes is enough for the N-bit parity case theoretically [41]. In this experiment, N-bit parity problems with N varying from 2 to 7 were used to test the effectiveness of the proposed algorithm. Figure 4.2 shows the number of hidden nodes generated by the progressive training for each of the parity problems. Each reading is an average of eight simulation attempts using different values of STEADYE(=30,50,150) and ζ(=2.5,10,100). It can be seen that the proposed algorithm has found the optimal number of hidden nodes, i.e. two, for the 2-bit parity (Exclusive-OR) problem. For the other cases, reasonably large networks though not optimal were obtained. All simulation trials were accomplished within 250 epoches except the 7-bit parity which took about 800 epoches. The standard BP trainings with fixed topologies were also performed but the results were so bad that a systematic comparison of the two algorithms was unnecessary. For example, BP spent more than 2000 epoches for a 12-hidden-node network to solve the 5-bit parity problem using

fine-tuned gain=0.05 and momentum=0.1. Larger gain or momentum would lead to local minimum cases, i.e., non-convergence. For 6-bit and 7-bit parity problems, none of the simulation trails converged.

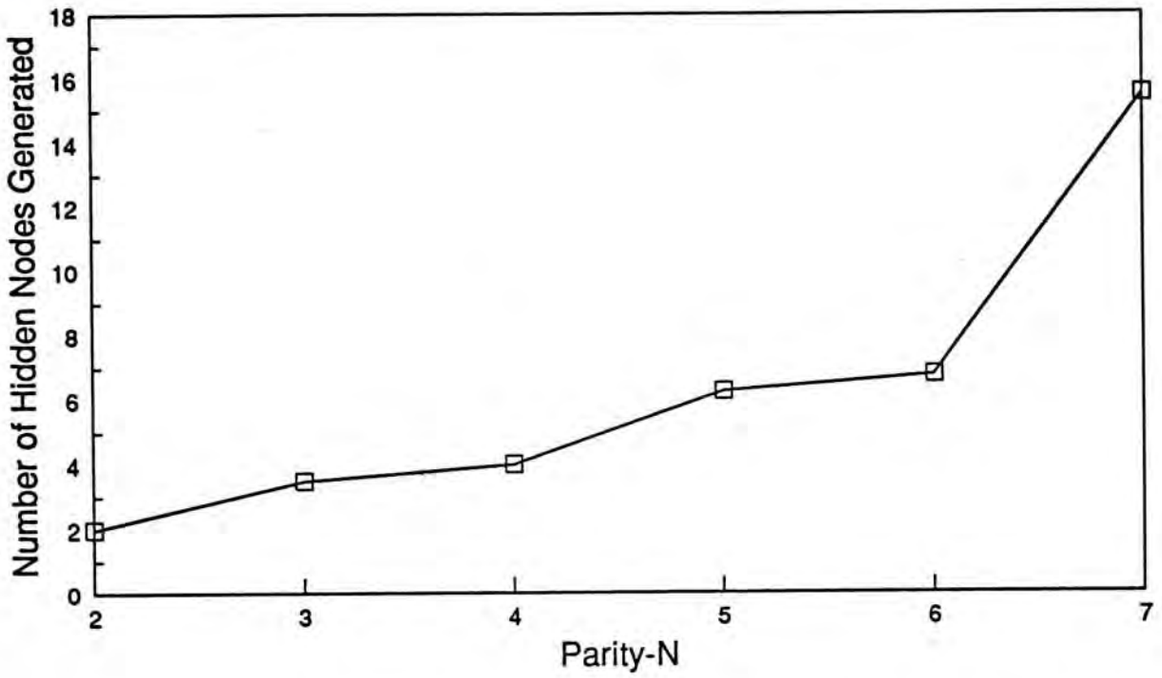

Figure 4.2 Number of hidden nodes generated by the progressive training : *N*-bit parity problems

(ii) IRIS Data Set

The IRIS data set of Fisher [56] has long been used to study the performance of various pattern classification algorithms. It has three pattern classes, two of which are overlapping. For each class, there are 50 samples and the dimension of which is four. A testing set was formed by randomly picking 30% of samples from each pattern class among the complete data set, the remaining 70% were used as testing data. Three training sets were made by randomly choosing 10, 40, and 70 out of the 70% training samples respectively. Thus they contained 15, 60, and 105 samples respectively, and the testing set consisted of 45 samples. All data points shown here were recorded as an average of eight readings obtained by using different values of STEADYE(=30,50,150) and ζ(=2.5,10,100) for progressive training and by using different number of hidden node (5,10,50) and initial weights for BP. Figure 4.3

shows the generalization performances of progressive training and standard BP with regard to the three training sets. As expected, training with more patterns generally classifies better on the testing data set. Both algorithms generalized well but the progressive training slightly outperformed BP in this case. In Figure 4.4, the number of hidden nodes generated for each training set (including the one with 100% of samples) by the progressive training is plotted. The trajectory shows that the network size increases as the training data set gets large and this illustrates the effectiveness of the proposed network growth algorithm in constructing networks that adapt to the complexity of the training data. The learning speeds of progressive training and BP are compared in Figure 4.5. To our surprise, progressive training is much faster than BP in this experiment. This is not the case in the handwriting numeral recognition experiment which will be reported later on. As mentioned previously, the IRIS data set is characterized by overlapping data which is quite usual in real world problems, and BP is believed to be very sensitive to this situation. In fact, BP training could not reach a solution for the last two training sets after 3000 epoches, probably getting stuck in local minima.

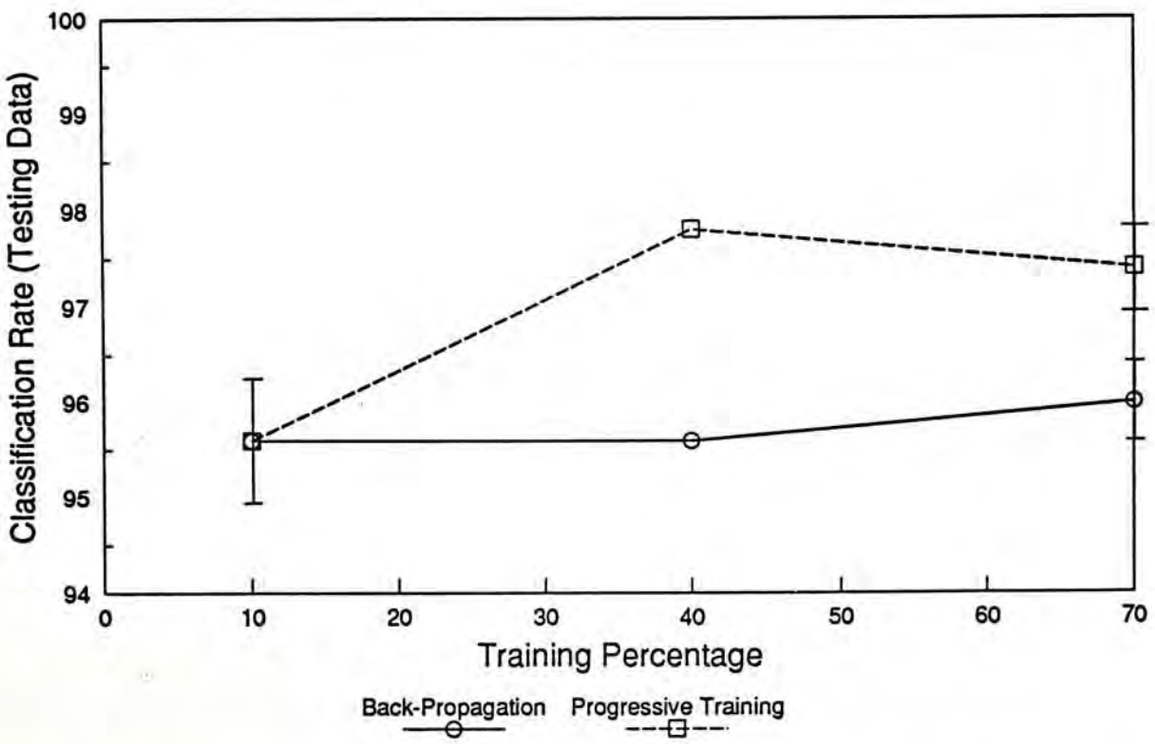

Figure 4.3 Generalization performance comparison of standard BP and progressive training : the IRIS data training sets

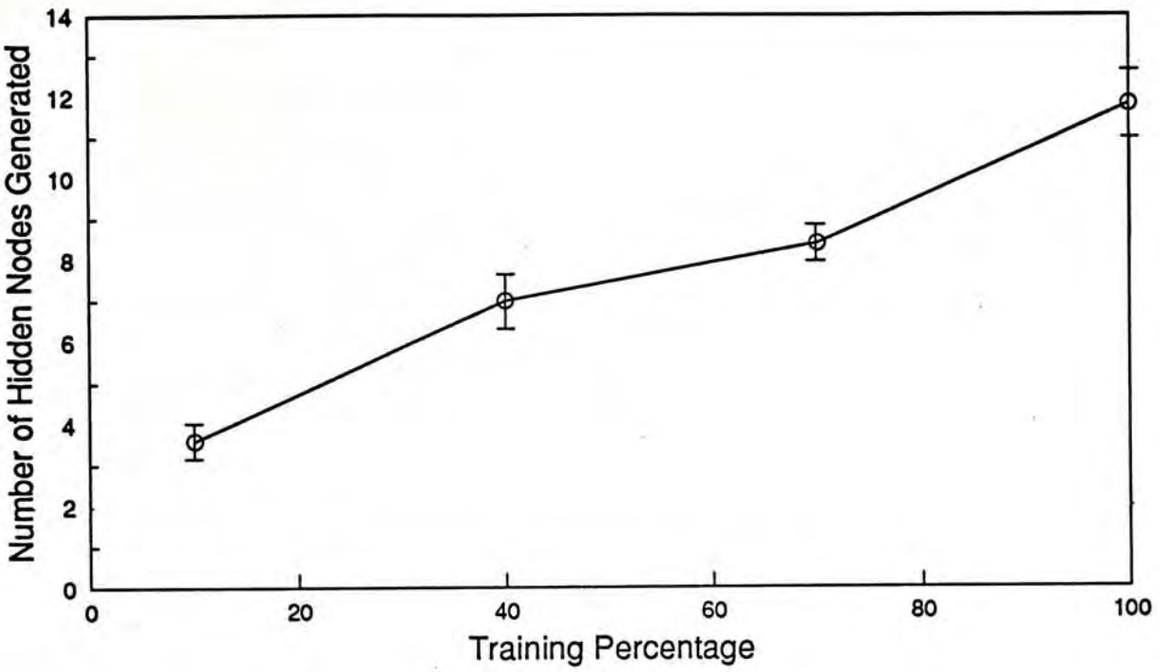

Figure 4.4 **Number of hidden nodes generated by the progressive training : the IRIS data training sets**

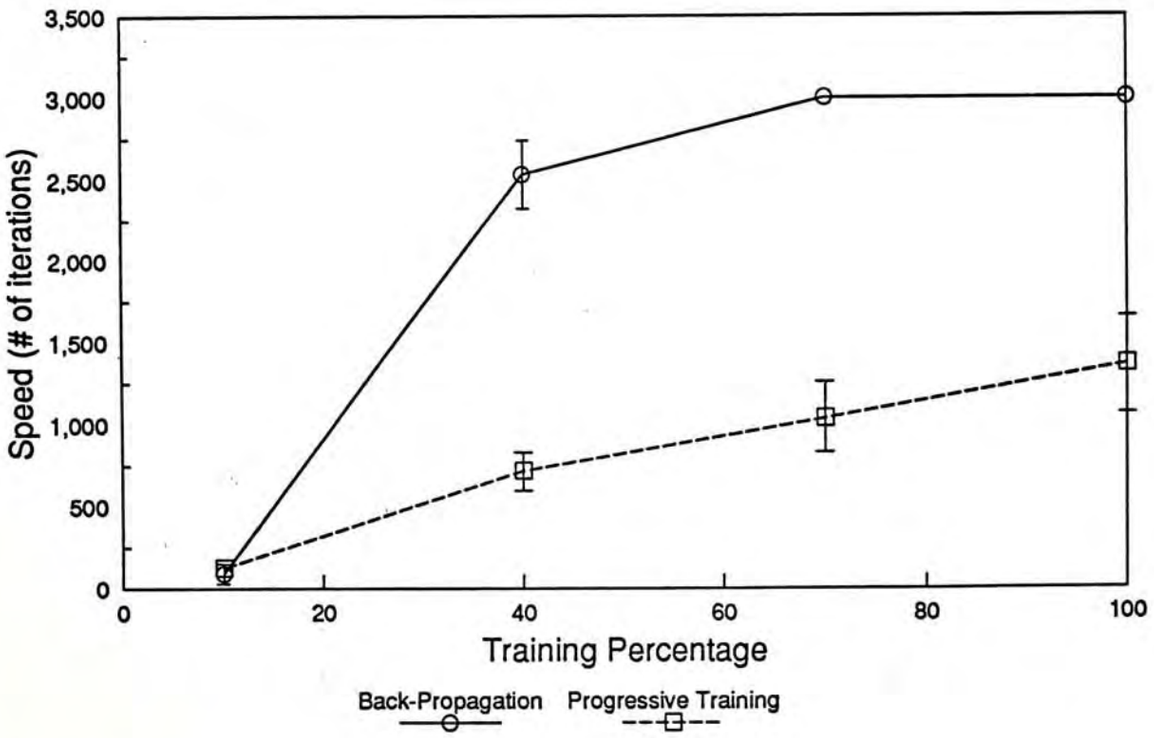

Figure 4.5 **Learning speed comparison of standard BP and progressive training : the IRIS data training sets**

(iii) Handwriting Numeral Recognition

The proposed algorithm was further evaluated with its application to a handwriting numeral recognition experiment. The feature extraction technique and sampling process of this experiment are depicted in Appendix A. The data set is composed of 30 samples for each of the ten numerals, each of which is represented by eight features. Thus there are 300 samples of ten pattern classes. Simulations performed here are the same as that of the IRIS data set and the results are recorded in Figures 4.6-4.8 respectively. Again, high generalization performances of the two algorithms are observed in Figure 4.6, however, this time progressive training is slightly inferior to BP. Unlike the IRIS data set, the node creation trajectory in Figure 4.7 is quite flat since inspection shows that the distribution of the ten clusters of numeral patterns are quite discrete, and therefore, the complexities of the four training sets are similar to each other and hence the numbers of hidden nodes generated by progressive training are close. Figure 4.8 shows that the learning speed of progressive training is slower than that of BP. In fact, the algorithm is slow because every time a new pattern is added to the current training subset, trainings have to go through all the previously learned patterns but only some of which require weight updating and hence many training epoches are wasted. Suggestions to remedy this drawback will be given in the concluding chapter.

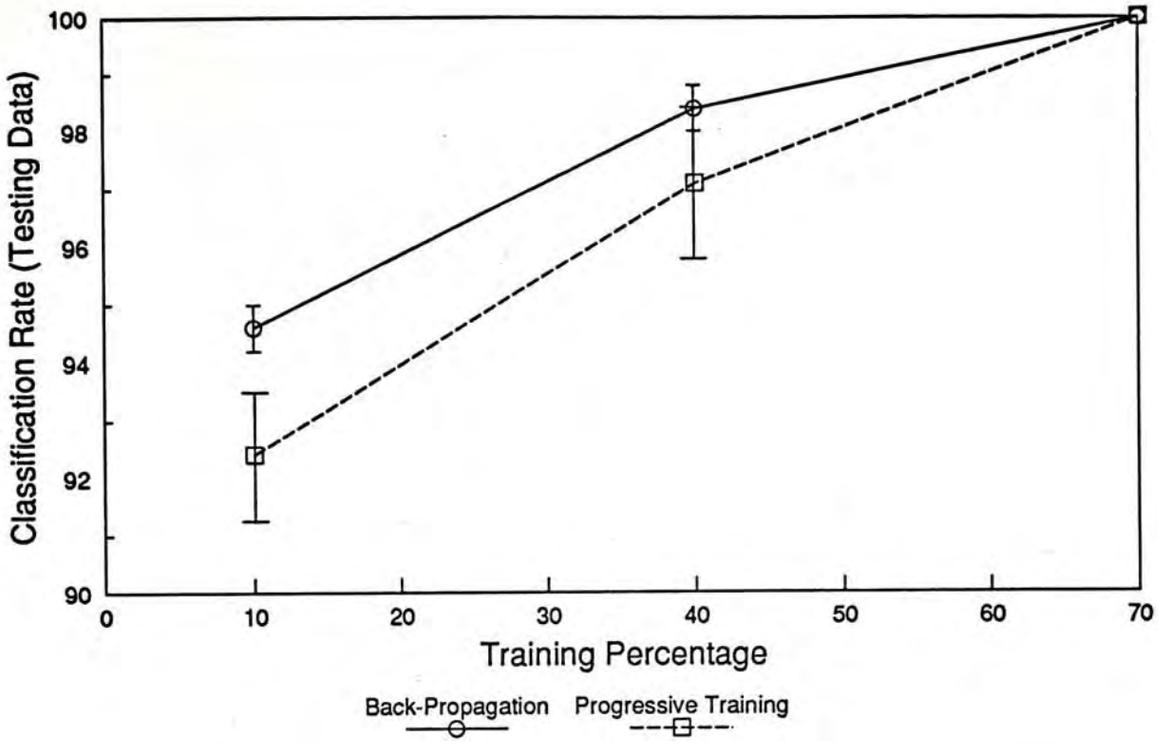

Figure 4.6 **Generalization performance comparison of standard BP and progressive training : the handwriting numeral training sets**

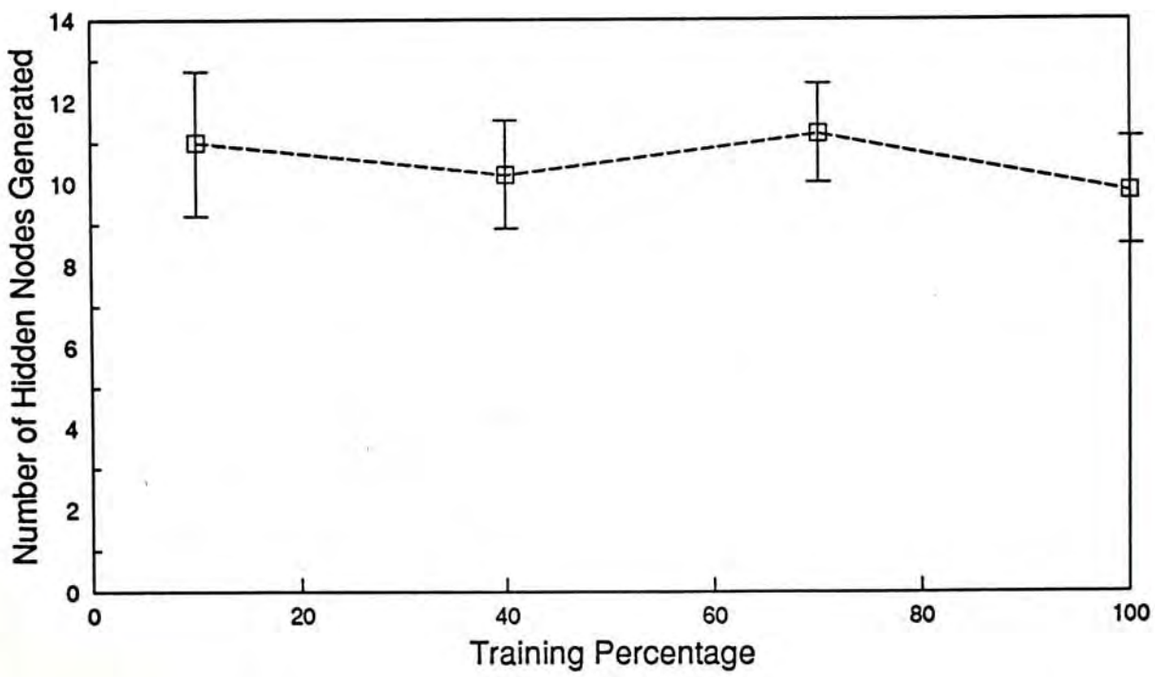

Figure 4.7 **Number of hidden nodes generated by the progressive training : the handwriting numeral training sets**

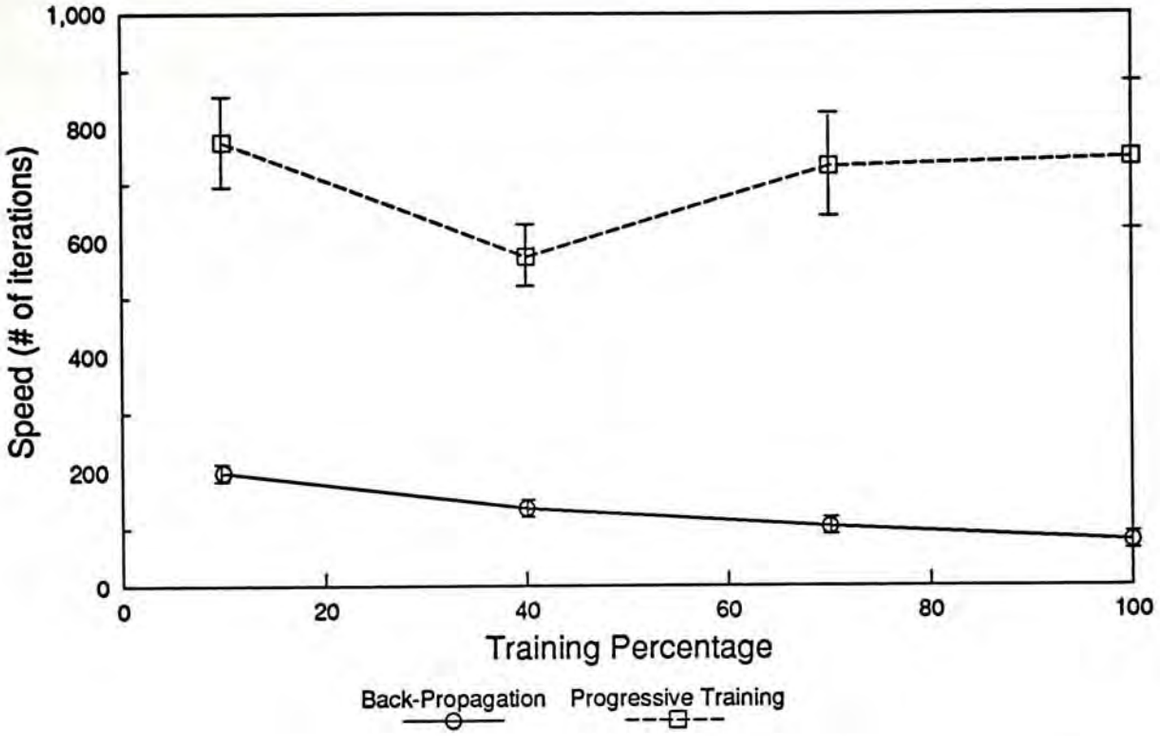

Figure 4.8 Learning speed comparison of standard BP and progressive training : the handwriting numeral training sets

(iv) Sensitivity Analysis of Parameters

The sensitivities of parameters scale-up factor ζ when STEADYE fixed at 50, and STEADYE when ζ fixed at 10 to the performance of progressive training in the 40% training set are depicted in Table 4.1 & 4.2 respectively. The data recorded are the average of eight simulation trails using different gain and momentum values. As expected, the scale-up factor has direct impact on the generalization performance of the algorithm. As ζ increases, the sigmoid function of the new hidden node tends to behave like a hard limiter which produces two output values only and hence its flexibility would be constrained. If unseen patterns are presented to the trained network, binary hidden (or internal) representation of those patterns will be resulted and this degrades the generalization ability of the network. Intuitively, if ζ gets large, the number of hidden nodes generated will increase and faster learning speed will result. It is because larger ζ values would cause the learning process to be more deterministic, in other words, the required network is formed by simply assigning values to new

nodes' weights and the number of hidden nodes generated tends to the theoretical upper bound stated in Theorem 1. In practice, the situation is slightly different. Table 4.1 shows that in the handwriting numeral recognition experiment, the learning speed with $\zeta = 100$ is the slowest. This is because more nodes were generated also and this in turn required more epoches to decide whether a new node should be added.

As stated before, the parameter STEADYE is used to specify when a new node should be added and when the new node's weights should be scaled up. Therefore, if it is set to a small value, more nodes will be generated and the learning time will be shorter. This is supported by the experimental results as recorded in Table 4.2. It also shows that using larger STEADYE seems to have better generalization. In our experience, STEADYE=50 & ζ=10 should be an appropriate set of values to compromise all these factors.

Table 4.1 Sensitivity of scale-up factor ζ (STEADYE=50)

Scale-up Factor ζ	Classification Rate (Testing Data Set)	Number of Nodes Generated	Learning Speed (epoches)
A. IRIS Data Set			
2.5	97.1%	6.7	924
10	97.1%	6.7	567
100	96.3%	6.7	408
B. Handwriting Numeral Recognition			
2.5	97.0%	11	643
10	96.7%	10.7	544
100	94.4%	13.3	808

Table 4.2 Sensitivity of STEADYE (ζ=10)

STEADYE (epoches)	Classification Rate (Testing Data Set)	Number of Nodes Generated	Learning Speed (epoches)
A. IRIS Data Set			
30	97.1%	7	365
50	97.1%	6.7	567
150	97.1%	4.7	866
B. Handwriting Numeral Recognition			
30	95.6	13.3	639
50	96.7	10.7	544
150	98.9	5.7	833

4.6 Concluding Remarks

We have presented a new network growth algorithm called progressive training for constructing a BP network automatically for any given set of non-conflicting training data, i.e., no identical patterns belong to different categories. With respect to previous network growth algorithms, such as the tiling algorithm [39] and upstart algorithm [41], the proposed algorithm advocates a completely new way of addressing the problem and convergence is guaranteed for any desired continuous input-output mapping which is beyond the scopes of previous works. The algorithm is easy to implement (see Figure 4.1) and the reported experimental results have demonstrated its effectiveness in finding a reasonably large network though not optimal for different tasks. Moreover, the generalization performance of the constructed networks was comparable to that of the fixed networks trained by standard BP. With progressive training, consideration regarding how to initialize the weights is no longer required. As confirmed by the experimental results, progressive training do not have local minimum problems. In the worst case, it will find a network with the number of hidden nodes equal to the theoretical upper bound (see Theorem 1). In fact, the algorithm steers

away from local minimum regions by its node adding process which will provide more feasible descending paths for the training process. However, a major drawback of the proposed algorithm is that the learning speed is slow in general and this remains to be improved.

5 PRUNING OF BP NETWORKS

5.1 Introduction

In this chapter, we switch our attentions to network pruning. Unlike network growth, network pruning attains an appropriate network by starting with an oversized network such that the probability for the BP to converge to a solution is higher. Unnecessary nodes and links in the converged oversized network are then removed afterwards. As mentioned in the introductory chapter, the difficulty of this approach is to identify the unnecessary nodes and links to be removed while the network performance would not be significantly impaired due to the removals. In contrast to previous works, the pruning algorithm proposed here removes nodes according to their "excessiveness" rather than their error contribution. This property is essential in determining whether a node can be removed with preserved network performance and hence the proposed algorithm is useful in obtaining a network that is optimized with both the network size and performance.

In this chapter, we report a study on the characteristics of hidden nodes in oversized network from the viewpoint of pattern classification theory and based on those findings, four categories of excessive nodes are identified as suitable candidates for pruning. An algorithm is subsequently proposed for attaining an appropriate size for a BP network by pruning excessive nodes. The pattern space and weight space interpretations of BP networks described in Chapter 3 will be used interchangeably in this work. In the next section we start by describing the characteristics of hidden nodes in oversized networks observed from an empirical study. Four categories of excessive nodes are then identified from the results and an insight to why they would be suitable candidates for pruning is also provided. The fourth section describes the proposed node pruning algorithm and the experimental results are reported in the following section. The chapter ends by a section devoted to concluding remarks.

5.2 Characteristics of Hidden Nodes in Oversized Networks

5.2.1 Observations from an Empirical Study

The characteristics of hidden nodes in oversized networks were investigated through an empirical study. Different oversized networks were used to solve various problems such as Exclusive-OR, 3-bit parity, and 2-D model data sets. Observations have been made from this study and they are summarized in the following.

(i) Under various simulations, there exists a set or sets of hidden nodes that are always included by the oversized networks. For example, Figure 3.1(a) showed one set of such nodes found for the Exclusive-OR problem.

(ii) Hidden nodes other than those mentioned in (i) are found to be arbitrary located in the input pattern space.

(iii) Further investigations on these two kinds of hidden nodes were carried out by pruning either kind of nodes from the trained networks and then further training was allowed. It was observed that pruning of the latter one could always be retrained to restore the original performance. However, pruning of the former one usually takes a long retraining time and sometimes could not be retrained to restore the original performance.

From the observations, we postulate that the hidden nodes would converge to one of the two kinds after training; they are solution nodes and excessive nodes. Solution nodes are those defined in (i) above and they cannot be removed from the network without significantly impairing its performance. Excessive nodes are those defined in (ii) above and they are the suitable candidates for pruning. These observations are consistent with the pattern space interpretation of BP networks described in Chapter 3. Since solution nodes will generate linearly separable hidden patterns, addition of an excessive node will just enlarge the dimension of the hidden patterns which will remain linearly separable. Hence the excessive nodes can be located everywhere in the input pattern space without affecting the linearly separability criterion. Thus if the excessive nodes can be identified, node pruning could be accomplished

in a straight forward manner. Before proceeding to derive the detection rules for excessive nodes, the characteristics of different excessive nodes will first be discussed because as a result different schemes were used to detect different kinds of excessive nodes.

5.2.2 Four Categories of Excessive Nodes

By using the weight space representation introduced in Chapter 3, four categories of excessive nodes have been identified. In order to better illustrate the identification process, an 1-D example is brought into discussions. In the 1-D example, there were five patterns $(x_1, x_2, \bar{x}_3, \bar{x}_4, x_5)$ in the input pattern space with co-ordinates : -1, -0.5, 0, 0.5 and 1.0 as shown in Figure 5.1. Patterns $x_1, x_2 \& x_5$ belonging to class A would have target value >0.5, while class B patterns $\bar{x}_3, \bar{x}_4$ had target value <0.5. The weight space representation of this example was depicted in Figure 5.2(a). It could be observed that no single weight vector will correctly classify all patterns, therefore, they were linearly non-separable. The weight space was divided into ten regions, R1 to R5 and R1' to R5' where Ri & Ri' were opposite to each other. A sample decision hyperplane from each region (e.g. Hi from Ri and Hi' from Ri') was depicted in Figure 5.2(b) to show the dichotomy of each decision hyperplane. Two samples were extracted from both R1 & R1' because the w_1-axis itself partitions the weight space into two regions of different characteristics.

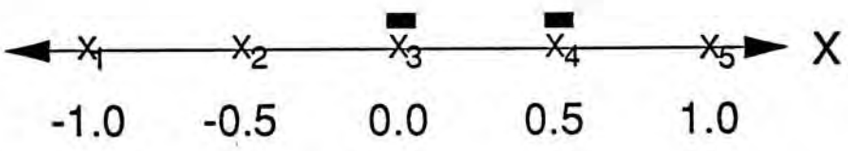

Figure 5.1 An 1-D classification example where patterns $x_1, x_2 \& x_5$ belong to class A and patterns $\bar{x}_3 \& \bar{x}_4$ belong to class B

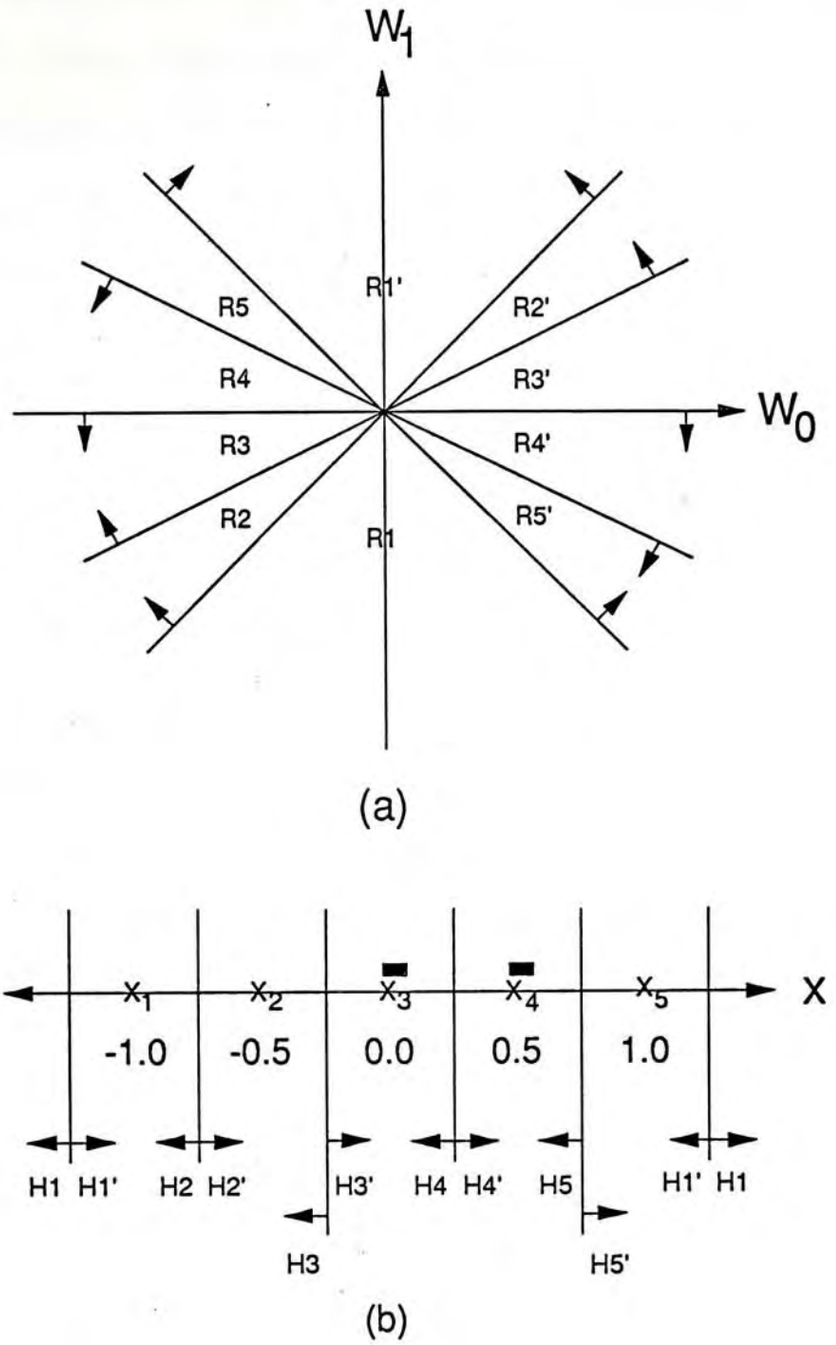

Figure 5.2 **(a) The weight space representation of the 1-D example consisting of ten regions (R1 to R5, R1' to R5'). (b) Decision hyperplanes in the input pattern space corresponding to each of the ten weight region with H3 & H5' denoting a set of solution nodes**

Empirical results showed that one set of solution nodes was located in the weight regions R3 & R5'. Hence, these two hidden nodes have generated linearly separable hidden patterns. As concluded from the previous section, excessive nodes can have weight vectors converged to any of the ten regions. However, weight vectors in different regions would have different properties and which would be explored in the following with the 1-D example illustrated in Figure 5.2. Notwithstanding the dimensionality of example used, the definitions and observations made for each category of excessive nodes are general.

1. Non-Contributing Node : If an excessive node has weight vector converged to one of the weight regions such as R1 or R1' in Figure 5.2(a), it is said to be non-contributing since it does not contribute in discrimination of the input patterns. The non-contributing node would give similar output response to all input patterns. For example, all the input patterns lie on one side of the decision hyperplanes of region H1 & H1' as illustrated in Figure 5.2(b).

2. Duplicated Node : If an excessive node has weight vector converged to the same weight region corresponding to one of the solution nodes (i.e., R3 or R5' in this case), it is said to be duplicated because it functions in a similar manner as one of the solution nodes.

3. Inversely-duplicated Node : If an excessive node has weight vector converged to the weight region which is opposite to that of a solution node (i.e., R3' or R5 in this case), it is said to be inversely-duplicated because it separates the input pattern space in the same manner as the solution node except that the responses are inverted.

4. Inadequate Node : If an excessive node has weight vector converged to either one of the weight regions R2, R2', R4 and R4', it is said to be inadequate because it functions partially as one of the solution nodes. For example, if one of the hidden node has decision hyperplane H5' to discriminate pattern x_5 from other patterns, the second solution node should have decision hyperplane such as H3 to discriminate $x_1 \& x_2$ from $\overline{x}_3 \& \overline{x}_4$. However, an inadequate node would have decision hyperplane such as H4 to discriminate $\overline{x}_4$ from $x_1, x_2 \& \overline{x}_3$. Therefore, H4 with H5' alone will not be adequate to completely separate the input patterns.

5.2.3 Why are they excessive ?

Based on the weight space representation, four categories of excessive nodes in the oversized networks have been identified. While the excessive nodes have been shown to yield trivial decision hyperplanes in the pattern space, their values of existence or rather, lack of value of existence, should be considered in the hidden pattern space domain because the hidden nodes are meant to produce linearly separable hidden patterns. In the following, we will investigate the effect of adding an excessive node on the distribution of hidden patterns. Again the 1-D example will be used for illustration. Figure 5.3 shows the 1-D hidden pattern space formed by the decision hyperplane H3 (a solution node). Obviously, they are not linearly separable and hence cannot be correctly classified by the output layer. If another hidden node is added and has decision hyperplane H5' which corresponds to another solution node, the 2-D hidden patterns will now be linearly separated by an output node L1 as shown in Figure 5.4. However, if the second node happens to be one of the excessive node, the hidden patterns will remain linearly non-separable. As for non-contributing nodes, the output states of this category of excessive nodes are either >0.5 or <0.5 for all the input patterns, and with the outputs close enough in magnitude, identical for instance, the 2-D hidden patterns would be formed by just augmenting the 1-D hidden patterns with a fixed component. A sample case for a non-contributing node located at H1' is shown in Figure 5.5(a). Obviously, the hidden patterns are not linearly separable. Since the duplicated nodes would have similar responses as those of the solution nodes for all patterns, two nodes would have very high correlation. As shown in Figure 5.5(b), the original 1-D hidden patterns would virtually be rotated by 45° in result and hence the 2-D hidden patterns are still linearly non-separable. If the second node is an inversely-duplicated node, then the two hidden nodes would have very high negative correlation. Thus, the 1-D hidden patterns would be mapped onto the dotted line shown in Figure 5.5(c) and the resultant 2-D hidden patterns are still linearly non-separable. It can be seen from Figure 5.3 that hyperplane H3 has divided the input patterns into two groups with one corresponds to $x_1 \& x_2$ and the other corresponds to $\overline{x}_3, \overline{x}_4 \& x_5$. Therefore,

the second hyperplane H5' should further separate the latter group such that the hidden patterns are linearly separable. Although the inadequate node follows this direction, it fails to completely separate pattern x_5 from patterns $\bar{x}_3\&\bar{x}_4$. A sample case for an inadequate node separating only $\bar{x}_3$ from $\bar{x}_4\&x_5$ is shown in Figure 5.5(d). The hidden patterns again are not linearly separable. However, if the hidden patterns are already linearly separable, addition of an inadequate node will not affect the distribution. Consider the same example with two solution nodes which has distribution of 2-D hidden patterns as depicted in Figure 5.4. The same inadequate node would further separate $\bar{x}_3$ from $\bar{x}_4$ this time, the new distribution is depicted in Figure 5.6. Obviously, the hidden patterns are still linearly separable. Note that the inadequate node only separates hidden patterns of the same output class.

Figure 5.3 Hidden patterns formed by a solution node

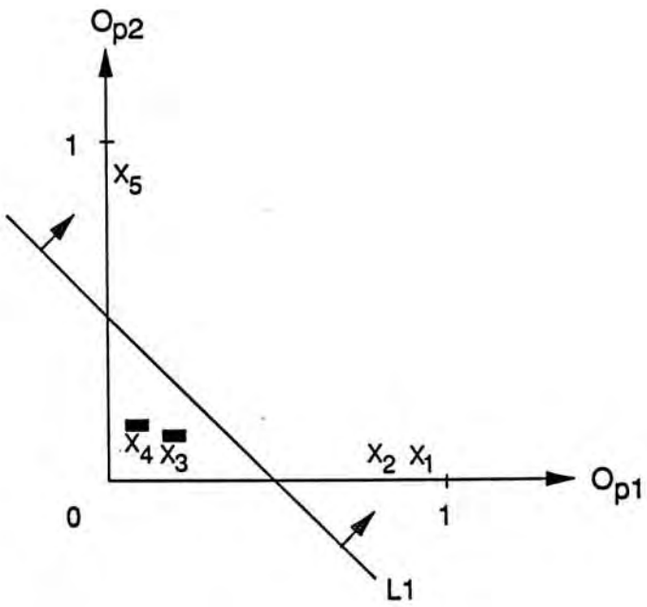

Figure 5.4 Hidden patterns formed by two solution nodes that can be separated by an output node L1

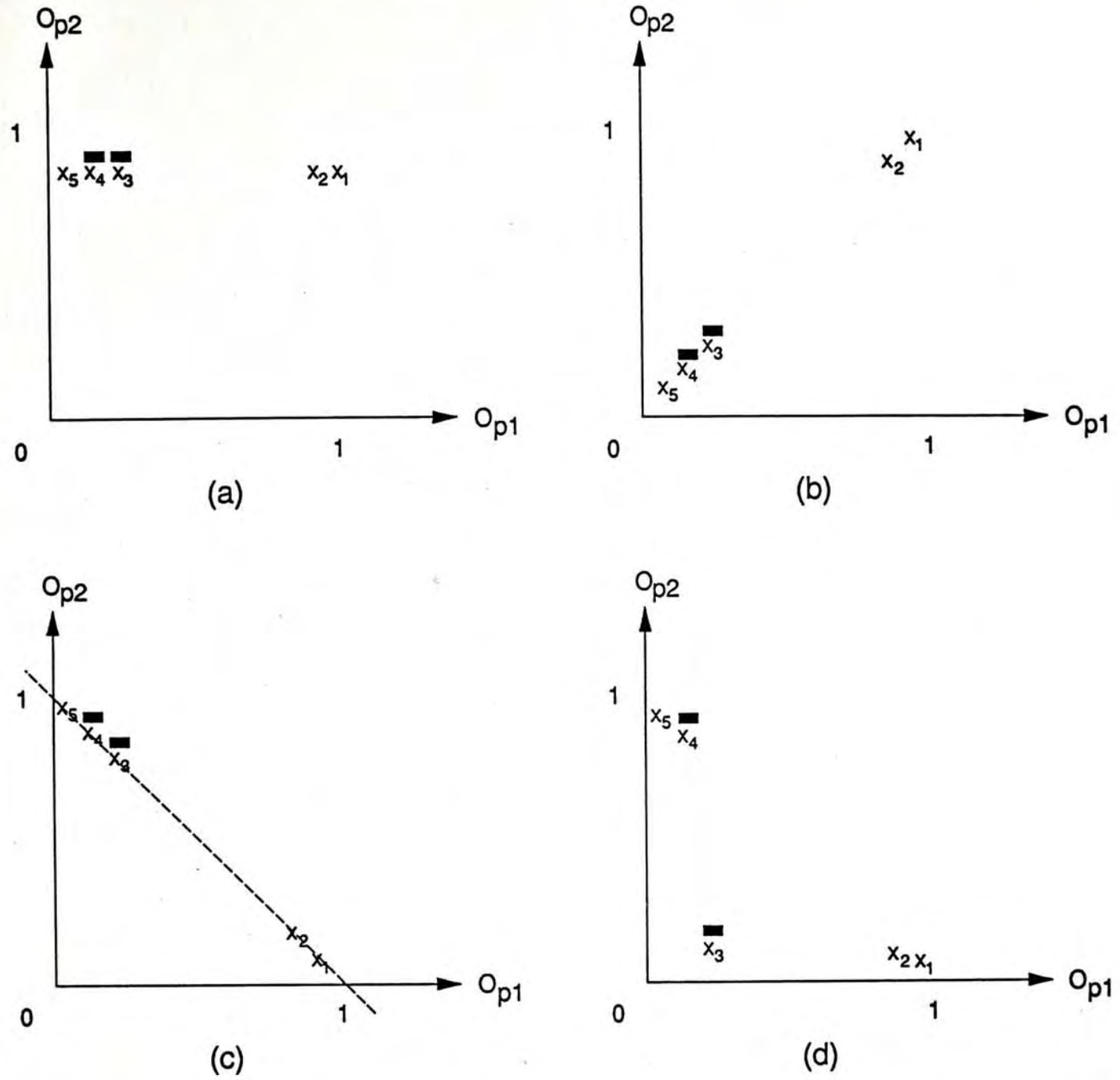

Figure 5.5 **Linearly non-separable hidden patterns formed by a solution and (a) a non-contributing node, (b) a duplicated node, (c) an inversely-duplicated node, and (d) an inadequate node**

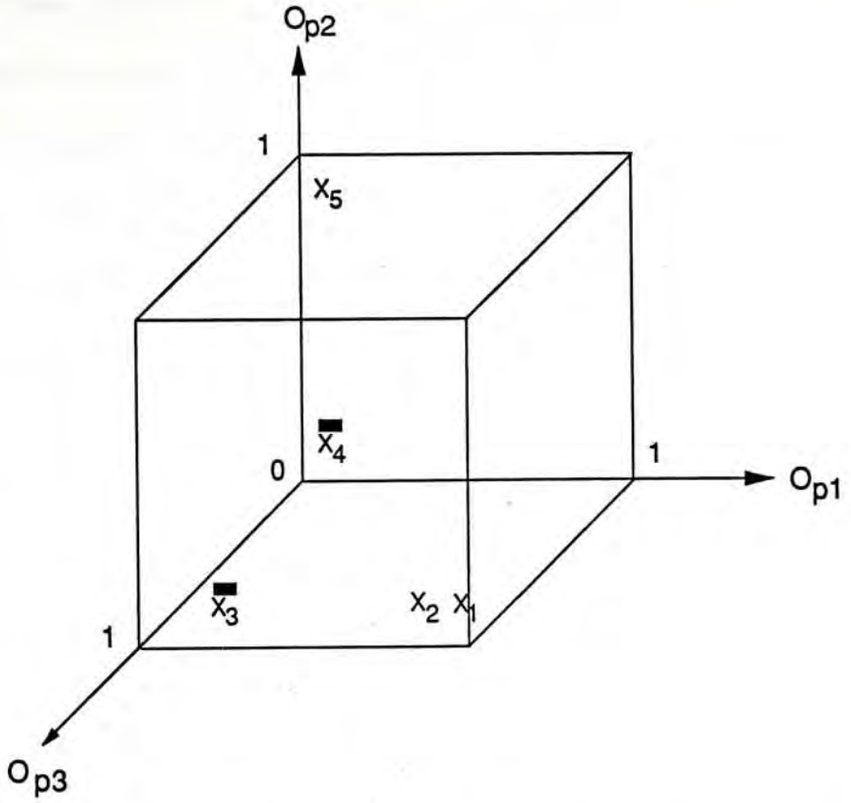

Figure 5.6 Linearly separable hidden patterns formed by two solution nodes and one inadequate node

5.3 Pruning of Excessive Nodes

In previous section, four categories of excessive nodes were identified and defined. Following their definitions, the detection rules for each category of excessive nodes were derived and based upon which an excessive node pruning algorithm was proposed.

(i) Detection Rule for Excessive Non-Contributing Nodes

Since a non-contributing node would have a decision hyperplane that places all input patterns to one side in the pattern space, this condition is first detected, viz.

Non-contributing = $(o_{pj} \geq 0.5 \ \forall p)$ OR $(o_{pj} < 0.5 \ \forall p)$

This condition alone, however is not sufficient for detecting excessive non-contributing nodes because we have showed, in last section that a non-contributing node is excessive and would not help to generate linearly separable hidden patterns when its outputs are close in magnitude for all input patterns. Thus, the following rule was proposed to detect an excessive non-contributing node.

$$\text{Excessive\_Non-contributing} = \text{Non-contributing AND } [\text{CLOSENESS}(o_{pj}) \leq \varepsilon_1]$$

where the CLOSENESS function returns a value in the range 0 - 0.5 and measures the closeness of o_{pj}'s magnitudes. If a small threshold is specified in the detection rule, only those non-contributing nodes adding almost fixed value component to the hidden patterns are detected. If large threshold, such as 0.5, is specified instead, then just non-contributing nodes will be detected. Therefore, if the above rule is used to identify potential nodes for pruning, adopting small threshold will identify less nodes to be pruned and will tend to preserve the performance of the original network after pruning. If a large threshold is adopted instead, the converse is true. Therefore, the threshold determines, to a certain extent, the level of trade-off between network size and network performance.

(ii) Detection Rule for Excessive Duplicated Nodes

Since a duplicated node duplicates at least one other node, if two nodes are found to discriminate the input patterns in a similar manner, one of them will be duplicated. Thus

$$\text{Duplicated} = (o_{pi} - 0.5)(o_{pj} - 0.5) > 0 \quad \forall p$$

One of the duplicated node will be excessive if the outputs of the duplicated nodes completely duplicated each other, i.e. $o_{pj} = o_{pi} \quad \forall p$. Thus the excessive duplicated nodes would be detected by

$$\text{Excessive\_duplicated} = \text{Duplicated AND } [\text{DIFF}(o_{pi}, o_{pj}) \leq \varepsilon_2 \quad \forall p]$$

where the DIFF function returns a value in the range 0 - 0.5 that measures the difference between the outputs of the duplicated nodes. Obviously the threshold ε_2 determines the tolerance in the definition of excessive duplicated nodes. Adopting small threshold in the detection rule will identify those closely correlated nodes while larger threshold will relax the definition and includes more nodes to be identified. Therefore, when this rule is used for pruning, the threshold again will determine the level of trade-off between network size and network performance.

(iii) Detection Rule for Excessive Inversely-Duplicated Nodes

Since an inversely duplicated node is just an inverse of a duplicated node, the detection rule will be identical to that of duplicated nodes, after inverting one of the nodes, viz.

$$\text{Inversely\_Duplicated} = (o_{pi} - 0.5)(o_{pj} - 0.5) < 0 \quad \forall p$$

and

$$\text{Excessive\_Inversely-Duplicated} = \text{Inversely-Duplicated AND } [\text{DIFF}(\text{INV}(o_{pi}), o_{pj}) \le \varepsilon_3 \quad \forall p]$$

where $\text{INV}(x) = 1 - x$. The threshold ε_3 has similar properties as ε_2 for detecting excessive duplicated nodes and again, determines the tolerance in the definition of excessive inversely-duplicated nodes and also the level of trade-off between network size and network performance.

(iv) Detection Rule for Excessive Inadequate Nodes

Since the excessive inadequate nodes have been observed to made same class separation in the hidden pattern space, its detection rule was derived based on this observation. The 1-D example in Figure 5.2(b) will be used for illustration again. Removing one of the solution nodes, H3 for example, class A pattern x_2 and class B pattern $\bar{x}_3$ will be merged to have similar hidden pattern representation. Practically x_2 has been merged with $\bar{x}_3$ to form a

mix-class cluster in the hidden pattern space due to the removal of node H3. Alternatively, removing the inadequate node H4 will cause the merging of $\overline{x}_3 \& \overline{x}_4$ which are both class B patterns. Thus removal of excessive inadequate node should cause merging of some hidden patterns without forming mix-class clusters in the hidden pattern space. Two functions, SAME_MERGE and MIX_MERGE were then explored to detect excessive inadequate nodes. The SAME_MERGE function checked if removal of a hidden node will cause the merging of some hidden patterns while the MIX_MERGE function checks if mix-class cluster would exist. Thus

Excessive_Inadequate-Node(j) = (NOT MIX_MERGE(j)) AND SAME_MERGE(j)

where function MIX_MERGE was implemented as

Function MIX_MERGE(j)

IF ($p_1 \& p_2$ are of different classes) AND $(o_{p_1 j} - 0.5)(o_{p_2 j} - 0.5) < 0$ THEN

MIX_MERGE(j) = $[(o_{p_1 i} - 0.5)(o_{p_2 i} - 0.5) > 0$ AND DIFF$(o_{p_1 i}, o_{p_2 i}) < \varepsilon_4]$ $\forall i \neq j$

and function SAME_MERGE was of the form

Function SAME_MERGE(j)

IF ($p_1 \& p_2$ are of the same classes) AND $(o_{p_1 j} - 0.5)(o_{p_2 j} - 0.5) < 0$ THEN

SAME_MERGE(j) = $[(o_{p_1 i} - 0.5)(o_{p_2 i} - 0.5) > 0$ AND DIFF$(o_{p_1 i}, o_{p_2 i}) < \varepsilon_5]$ $\forall i \neq j$

Again, a DIFF function was used to measure how close the clusters would be after merging. Therefore the parameters $\varepsilon_4 \& \varepsilon_5$ determined the level of trade-off between network sizes and performance due to pruning of excessive inadequate nodes. With large ε_4 and small ε_5, the detection rule will be more stringent and favours in maintaining the performance.

(v) A Node Pruning Algorithm

A node pruning was then derived based upon the detection rules for different categories of excessive nodes. First they were detected from a network and pruned. After pruning, the network has to be retrained to obtain the proper weights for the output layer.

Five thresholds $\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4, \& \varepsilon_5$ have been introduced in the detection rules and the effects of these thresholds on the detection rules have been briefly discussed. In summary, these thresholds determined the level of trade off between network performance and network size when the detection rules were used for node pruning. The sensitivity of the thresholds to these two performance criteria have been summarized in Table 5.1. The thresholds should be chosen between tabulated limits to suit individual applications.

Table 5.1 Sensitivity of threshold parameters to pruning

Threshold					Sensitivity	
ε_1	ε_2	ε_3	ε_4	ε_5	Pruning Mechanism	Performance of pruned network
0.0	0.0	0.0	0.5	0.0	soft (prune fewer nodes)	not affected
0.5	0.5	0.5	0.0	0.5	hard (prune more nodes)	may be affected slightly

5.4 Experimental Results and Performance Analysis

In this section, the effectiveness of the proposed node pruning algorithm is demonstrated through the experiments in learning the IRIS data set and the handwriting numeral database. All simulations employed networks with one hidden layer. Trainings of the networks were performed using the standard BP and the implementation details are the same as that described in Chapter 4 for BP training.

(i) IRIS Data Set

Again, the IRIS data set was used to evaluate the proposed node pruning algorithm. Simulations were conducted using oversized networks of different sizes and the results were summarized in Table 5.2 with each reading recorded as the average of five simulation trails using different sets of random initial weights. For each initial network size, five sets of threshold parameters were chosen to simulate and arranged from hard to soft pruning in Table 5.2. The results were consistent with the parameter sensitivity analysis in Table 5.1 since with relatively hard pruning thresholds, say $\varepsilon_{1-3} = 0.5$, $\varepsilon_4 = 0.1$, and $\varepsilon_5 = 0.5$, the pruning algorithm would try to optimize the network size as much as possible. It can be observed that by using the first three sets of thresholds, networks of different initial size were finally pruned to the nearly optimal network, i.e., around four hidden nodes. On the other hand, with relatively soft thresholds such as $\varepsilon_{1-3} = 0.1$, $\varepsilon_4 = 0.5$, and $\varepsilon_5 = 0.1$, the algorithm pruned fewer nodes in result. This contrast was magnified when the initial number of hidden node was ridiculously large, i.e. 50. Another observation is that if the initial network size is not an oversized one, i.e. three hidden nodes, application of the proposed algorithm would not be harmful and this should be an essential property in network pruning. To our surprise, in all cases the network maintained or even improved the performance of the original oversized network after pruning. This was not the case in the handwriting numeral experiment.

Table 5.2 Node pruning simulation results of IRIS data set

Parameters			Number of hidden node		Classification Rate (%)	
$\varepsilon_1 - \varepsilon_3$	ε_4	ε_5	Before Pruning	After Pruning	Before Pruning	After Pruning
0.5	0.1	0.5	3	3	98.7	98.7
			5	4	99.0	99.3
			10	5	98.7	99.3
			50	5	98.7	98.7
	0.5	0.5	3	3	98.7	98.7
			5	4	99.0	99.3
			10	5	98.7	99.3
			50	5	98.7	98.7
	0.5	0.1	3	3	98.7	98.7
			5	4	99.0	99.3
			10	5	98.7	99.3
			50	6	98.7	98.7
0.3	0.5	0.1	3	3	98.7	98.7
			5	4	99.0	99.3
			10	7	98.7	99.3
			50	7	98.7	98.7
0.1	0.5	0.1	3	3	98.7	98.7
			5	4	99.0	99.3
			10	7	98.7	99.3
			50	14	98.7	98.7

(ii) Handwriting Numeral Recognition

The proposed node pruning algorithm was further tested on the handwriting numeral database described in Chapter 4. The simulation results were recorded in Table 5.3. Each reading is the averaged value of five simulation trails using different sets of initial weights. Again, more nodes would be pruned by using relatively hard thresholds and fewer nodes would be pruned by using relatively soft thresholds. Unlike the first experiment, the classification performance using the set of relatively hard thresholds $\{\varepsilon_{1-3} = 0.5,\ \varepsilon_4 = 0.1,$ and $\varepsilon_5 = 0.5\}$ was slightly inferior to that of the original oversized network when the initial

network consisted of ten hidden nodes. However, using the other sets of thresholds maintained the original high classification rate in the expense of pruning fewer nodes. Trade-off between pruned network's performance and the number of hidden nodes being pruned was illustrated in this experiment.

Table 5.3 Node pruning simulation results of handwriting numeral data set

Parameters			Number of hidden node		Classification Rate (%)	
$\varepsilon_1 - \varepsilon_3$	ε_4	ε_5	Before Pruning	After Pruning	Before Pruning	After Pruning
0.5	0.1	0.5	3	3	100	100
			5	3	100	100
			10	4	100	95
			50	12	100	100
	0.5	0.5	3	3	100	100
			5	5	100	100
			10	8	100	100
			50	17	100	100
	0.5	0.1	3	3	100	100
			5	5	100	100
			10	8	100	100
			50	19	100	100
0.3	0.5	0.1	3	3	100	100
			5	5	100	100
			10	8	100	100
			50	20	100	100
0.1	0.5	0.1	3	3	100	100
			5	5	100	100
			10	8	100	100
			50	29	100	100

(iii) Generalization Performance

To investigate the generalization performances of pruned networks, the three training sets (consisting of 10%, 40%, and 70% of the available training samples) and the testing set (having the other 30% of samples), described in Chapter 4, for the IRIS data set and the handwriting numeral database were employed in this simulation. Table 5.4 shows the generalization performances of the network before and after pruning for each of the six training sets.

Each reading in the column "Before Pruning" is an average of eight simulation attempts using different initial oversized networks (with 3,5,10 & 50 hidden nodes) and different sets of initial weights. For the column "After Pruning", each reading took on the average of twenty-four values since three representative sets of thresholds were used to prune each of the eight trained networks. The three sets of thresholds included $\{\varepsilon_{1-3}=0.5,\ \varepsilon_4=0.5,\ \varepsilon_5=0.5\}$, $\{\varepsilon_{1-3}=0.3,\ \varepsilon_4=0.5,\ \varepsilon_5=0.5\}$, and $\{\varepsilon_{1-3}=0.1,\ \varepsilon_4=0.5,\ \varepsilon_5=0.5\}$. In addition to the averaged values, the quantities of the standard derivation are recorded inside the brackets. From these results, it can be seen that the generalization performance before pruning is competitive with that after pruning. Since initial networks with 3,5,10 & 50 hidden nodes were all taken into accounts in Table 5.4 and this may smooth out the performance discrepancy between the oversized networks and pruned networks, Table 5.5 reports also on the results obtained by the 50-hidden-node oversized networks only but it does not make too much difference. In fact, after careful inspections, it can be observed that the generalization performance after pruning has been slightly improved for the IRIS data set. No such observation can be obtained from the handwriting numeral data set. As mentioned in Section 4.5 that the IRIS data set is characterized by overlapping data while the distribution of the ten clusters of numeral data is quite discrete, it is reasonable to have improved generalization performances from pruned networks for the IRIS data set but not for the handwriting numeral data set since classifying overlapping data would have a better reflection how good the regularity of training data was captured by the network.

Table 5.4

Generalization performance before and after Pruning

Training Percentage	Classification on Testing Set: Before Pruning	After Pruning
A. IRIS Data Set		
10%	95.6% (1.3%)	95.7% (0.2%)
40%	95.6% (0.0%)	95.6% (0.0%)
70%	95.9% (0.7%)	96.2% (1.0%)
B. Handwriting Numeral Recognition		
10%	95.0% (2.2%)	95.0% (2.2%)
40%	98.2% (0.9%)	97.8% (1.1%)
70%	100.0% (0.0%)	100.0% (0.0%)

Table 5.5

Generalization performance before and after pruning (50-hidden-node networks)

Training Percentage	Classification on Testing Set: Before Pruning	After Pruning
A. IRIS Data Set		
10%	95.6% (0.0%)	96.0% (0.4%)
40%	95.6% (0.0%)	95.6% (0.0%)
70%	96.7% (1.1%)	96.7% (1.1%)
B. Handwriting Numeral Recognition		
10%	94.4% (0.0%)	94.6% (0.2%)
40%	98.9% (0.0%)	97.7% (1.1%)
70%	100.0% (0.0%)	100.0% (0.0%)

5.5 Concluding Remarks

Through a study on the characteristics of hidden nodes in pattern space and weight space representation, an insight to the properties of excessive hidden nodes were obtained and formally defined. Since the excessive nodes have been shown to have little contributions in producing the network performance, it is theoretically justified to propose a node pruning algorithm to remove the excessive hidden nodes from an oversized network while preserving the network performance such that the pruned network is more efficient to implement. The reported experimental results demonstrated the effectiveness of the proposed node pruning algorithm in reducing a network size without significantly impairing its original classification accuracy.

A pre-requisite of the proposed algorithm is, however, the existence of an initial BP network that already has reasonable performance. The initial network may have obviously excessive number of hidden nodes and may be trained by standard BP algorithm or its modified version [19-22]. If a training algorithm generates a solution for a given problem with optimal number, or close to that, of hidden nodes, application of the proposed algorithm would not be beneficial but not harmful, an essential property of network pruning algorithms. Otherwise the proposed algorithm can be implemented as a post-process to obtain a network that is optimized with respect with both the network size and classification performance. Although five parameters have been introduced in the proposed pruning algorithm, the experimental results showed that the thresholds would only fine-tune the trade-off between network size and accuracy while all parameter settings gave satisfactory results on node pruning.

6 DYNAMIC CONSTRUCTION OF BP NETWORKS

6.1 A Hybrid Approach

We have addressed the network design problem from two different approaches, i.e., network growth and network pruning, and presented a progressive training algorithm and a node pruning algorithm respectively. The progressive training starts from scratch to construct a reasonably large network gradually for the available training data set. Hence intelligent guess on the network size is not necessary. However, the constructed network might not be an optimal one and some sorts of non-essential nodes and links might involve. As mentioned at the very beginning of the thesis, such kind of networks suffers from degraded generalization performances and inefficient hardware realizations. The node pruning on the other hand assumes the existence of an initial oversized network that is well-trained and unnecessary nodes and links are then removed to attain a network with optimized size and performance. In other words, one has to predetermine an initial oversized network structure, and this creates a need for the reliance of the network designers on *a prior* knowledge about the task such as its complexity. Thus, it is reasonable to consider a hybrid approach which takes advantages of individual approaches. In fact, the two proposed algorithms are ready to combine. By simply cascading the node pruning process to the progressive training, a new algorithm is formed. Figure 6.1 shows the flowchart of this hybrid algorithm. It can be seen that the first part of the algorithm is the same as the progressive training as depicted in Figure 4.1, through which hidden nodes are added one by one to an initial one-hidden-node network in order to adapt to the complexity of the available training patterns. The constructed network would be an appropriate oversized network to apply the second part of the algorithm, i.e., node pruning. The four categories of excessive nodes are detected and then pruned from the network. After pruning, the network is retrained to obtain an optimal network that is minimal in size and is expected to have good performance on both the training and unseen data.

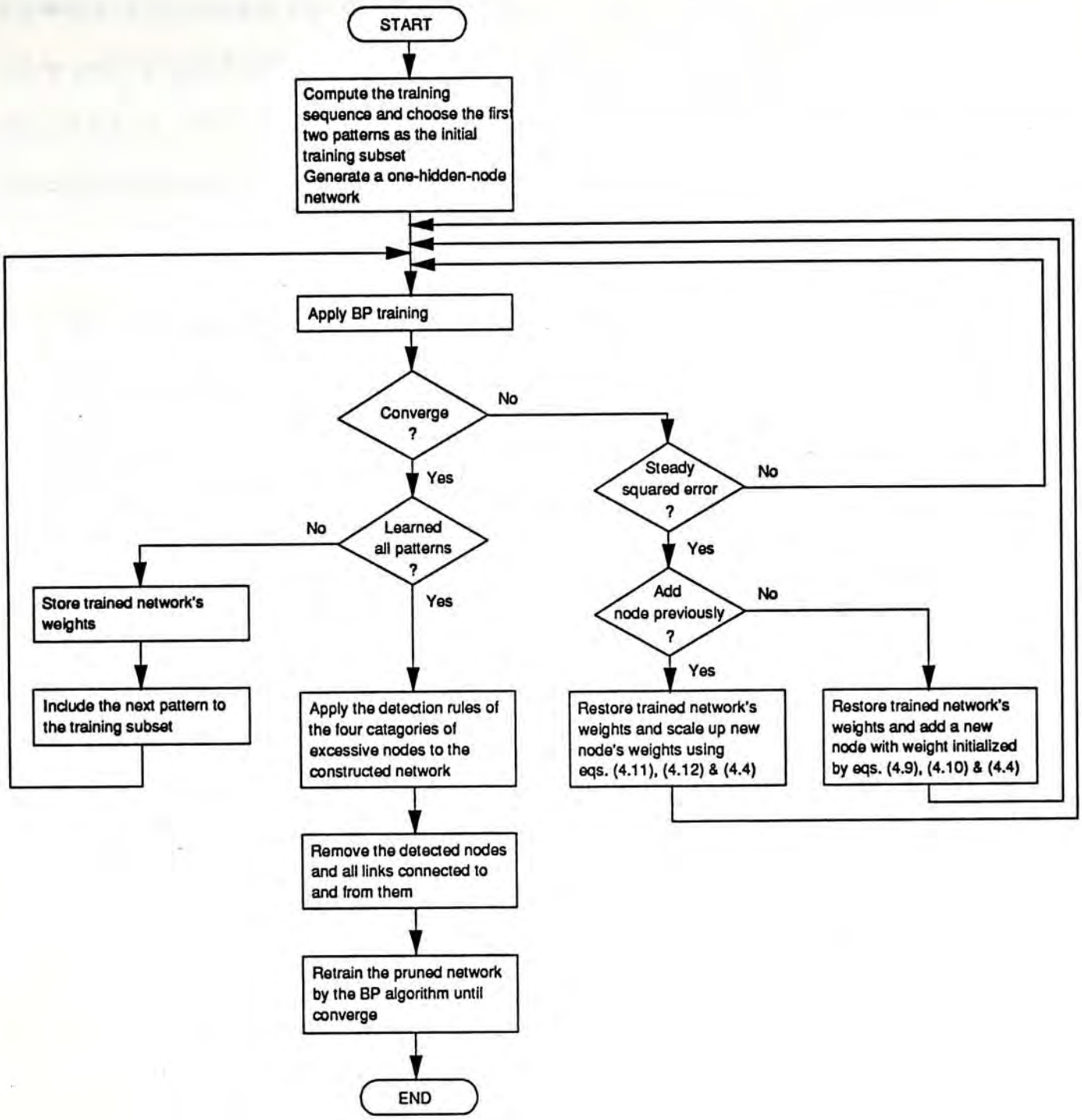

Figure 6.1 Flowchart of the hybrid algorithm

6.2 Experimental Results and Performance Analysis

Since the proposed hybrid algorithm extends the progressive training by incorporating the node pruning algorithm as a post-process to obtain a more constrained network, the experimental results reported in this section are simply a continuation of those presented in Chapter 4. In other words, the implementation detail regarding the first part of the hybrid

algorithm is the same as that described before. For the node pruning part, the three representative sets of thresholds, i.e., $\{\varepsilon_{1-3}=0.5,\ \varepsilon_4=0.5,\ \varepsilon_5=0.5\}$, $\{\varepsilon_{1-3}=0.3,\ \varepsilon_4=0.5,\ \varepsilon_5=0.5\}$, $\{\varepsilon_{1-3}=0.1,\ \varepsilon_4=0.5,\ \varepsilon_5=0.5\}$ were employed throughout this simulation and only the averaged values would be recorded in the following figures.

(i) N-bit Parity

The performance of the hybrid algorithm, comparing with those of the standard BP and progressive training, is shown in Figure 6.2. It can be observed that for N<7 the progressive training has already found the nearly optimal networks, and in these cases significant benefits could not be obtained by the hybrid algorithm. However in case N=7, the size of the network has been cut down by a quarter of magnitude.

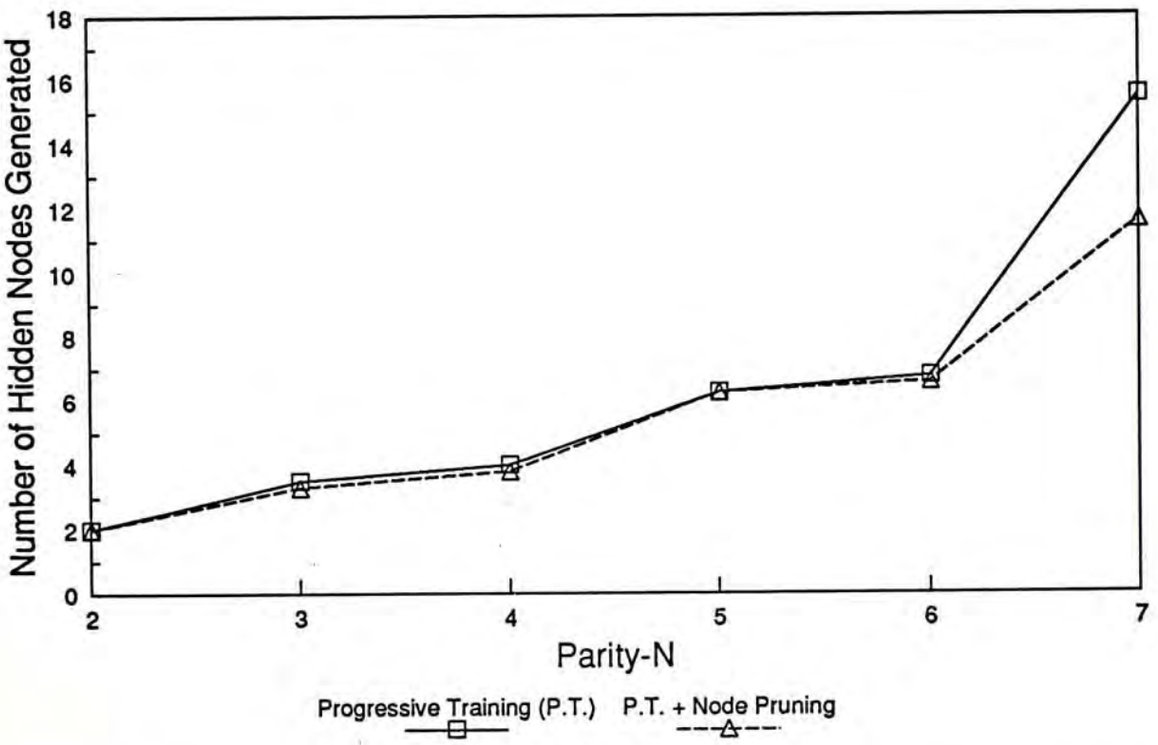

Figure 6.2 Number of hidden nodes generated by the hybrid algorithm : *N*-bit parity problems

(ii) IRIS Data Set and Handwriting Numeral Recognition

In the experiments with the IRIS data set and the handwriting numeral database, the effectiveness of the hybrid approach is fully demonstrated. Figures 6.3 & 6.4 depict the size of the finalized networks found by the hybrid algorithm. Again the original network size has been reduced by approximately 30% in average. Besides, the generalization performances of the finalized networks were compared with those of the fixed topologies trained by standard BP and the networks constructed by progressive training. The results were recorded in Figures 6.5 & 6.6 for the IRIS data set and handwriting numeral database respectively. It can be observed that the performances of the finalized networks are slightly inferior to those of the other two but the difference is negligible.

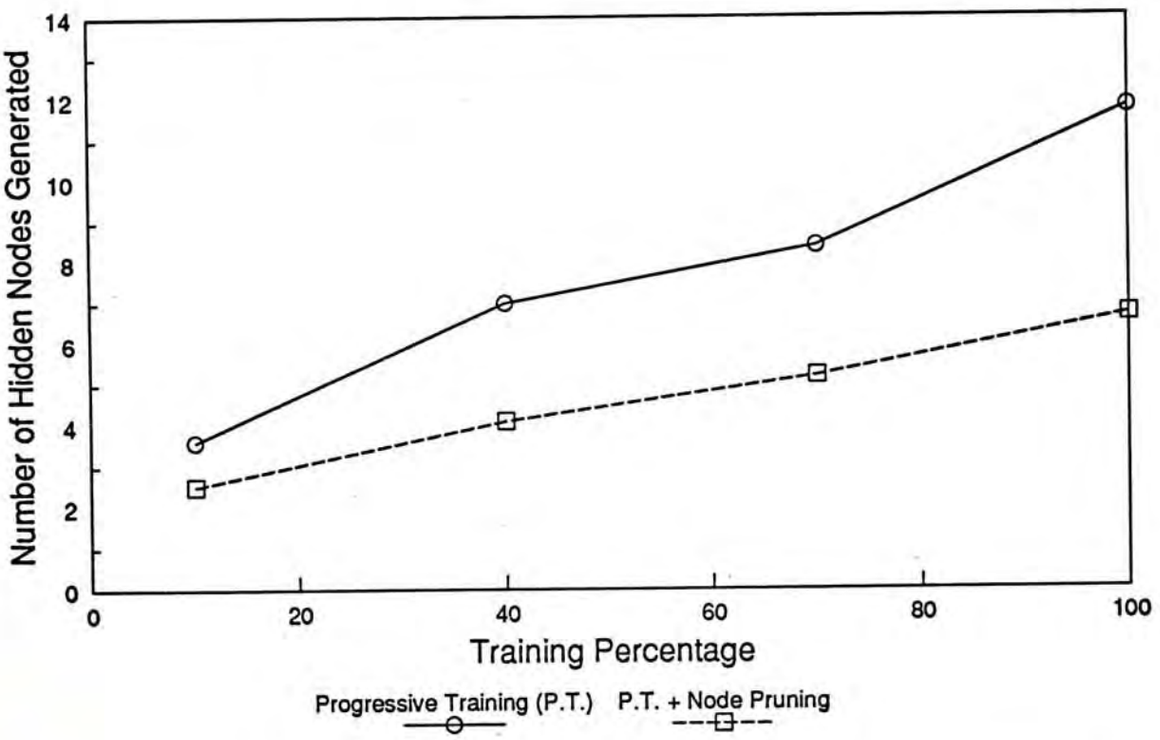

Figure 6.3 Number of hidden nodes generated by the hybrid algorithm : the IRIS data set

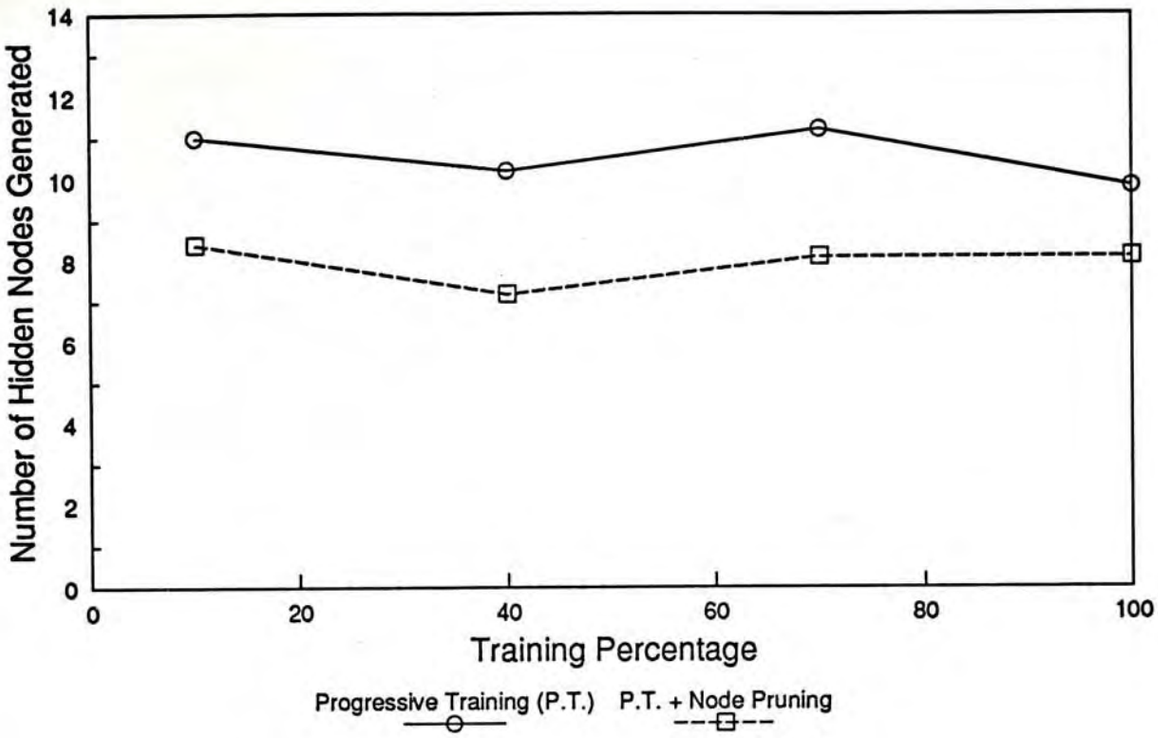

Figure 6.4 **Number of hidden nodes generated by the hybrid algorithm : the hand-writing numeral data set**

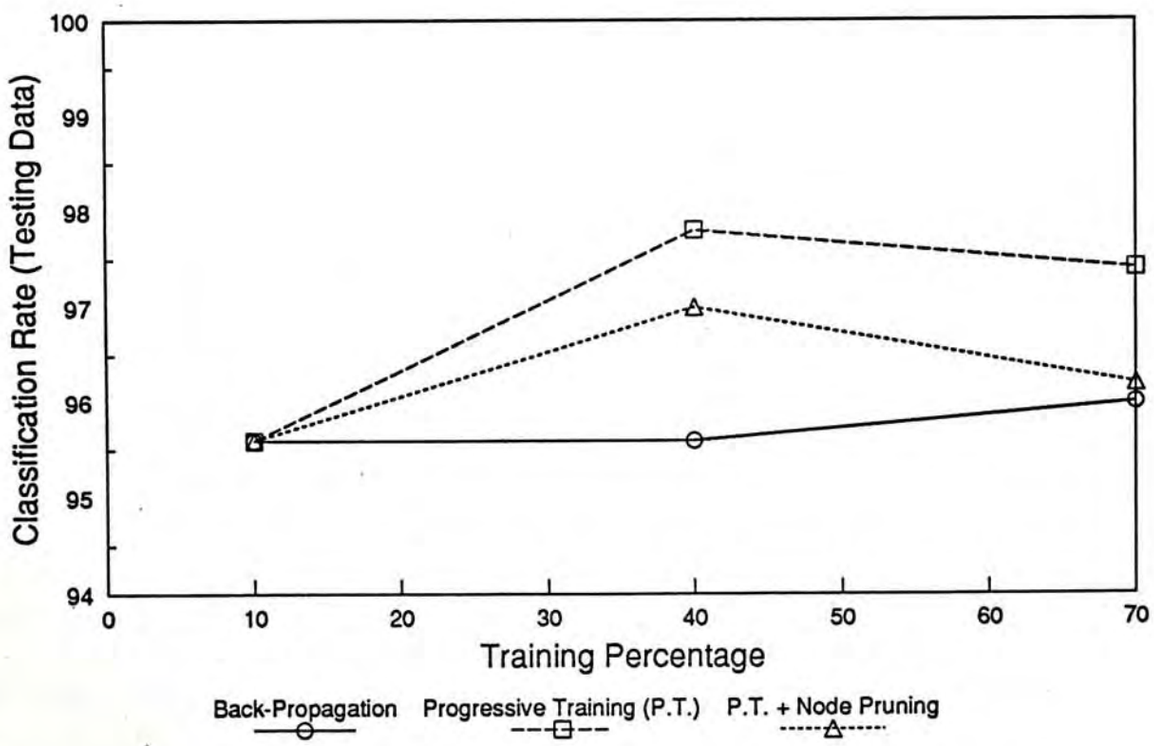

Figure 6.5 **Generalization performance of the hybrid algorithm : the IRIS data aet**

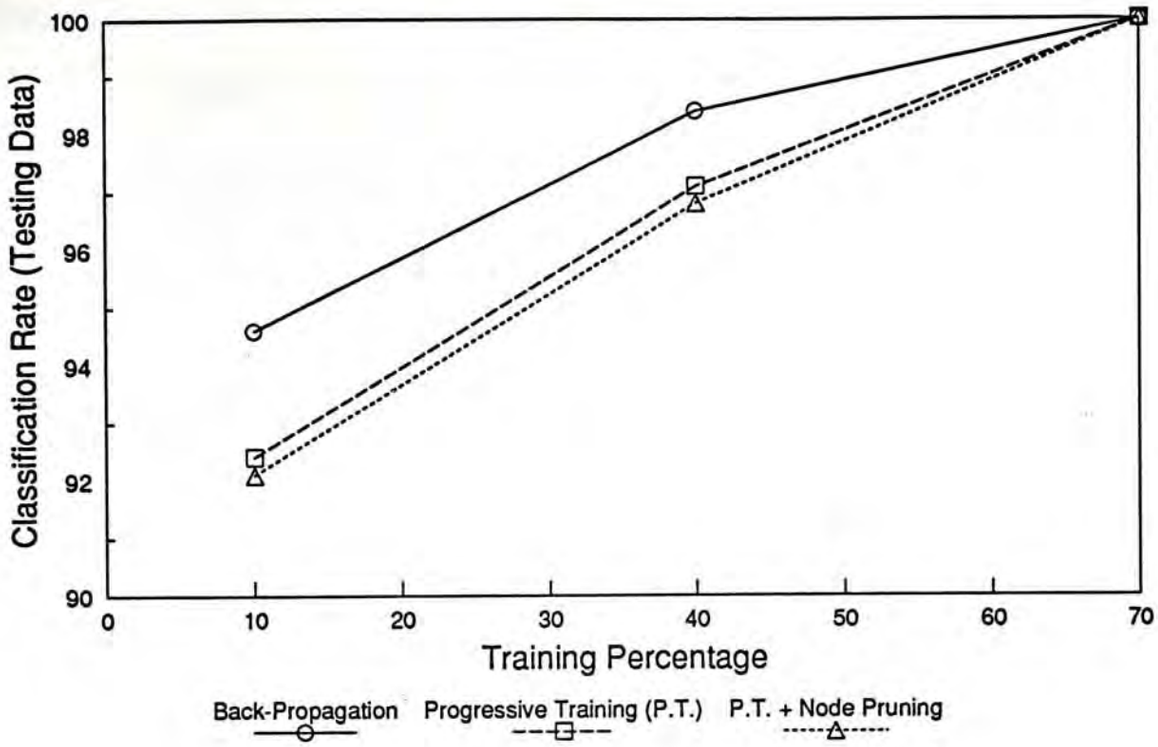

Figure 6.6 Generalization performance of the hybrid algorithm : the handwriting numeral data set

(iii) A Note on Specifying Parameter Values for the Hybrid Algorithm

In this subsection, we make a note on how the parameter values of progressive training and node pruning in the hybrid algorithm are chosen. Table 6.1 depicts a sample network construction process using different sets of parameter values for the 40% training sets of IRIS data set and handwriting numeral data set. By using a small value for STEADYE(=30), more nodes were generated by the progressive training part, and the pruned network sizes corresponding the three sets of threshold parameters for relatively soft, medium, and hard pruning are quite different. On the other hand, by using a large value for STEADYE(=150), networks with fewer nodes were obtained and the influence of the threshold parameter setting is relatively lower. Therefore, if one would like to have fast training by using small STEADYE value (see the parameter sensitivity analysis of progressive training in Section 4.5), hard pruning is suggested to obtain a smaller network. However, in case STEADYE is large, any degree of pruning would not make too much difference.

Table 6.1 Network construction process using different sets of parameter values

After progressive training			and then after node pruning			
STEADYE	ζ	Number of hidden nodes generated	$\varepsilon_1 - \varepsilon_3$	ε_4	ε_5	Number of hidden nodes
A. IRIS Data Set						
30	10	9	0.5	0.5	0.5	4
			0.3	0.5	0.5	5
			0.1	0.5	0.5	7
150	10	5	0.5	0.5	0.5	3
			0.3	0.5	0.5	4
			0.1	0.5	0.5	4
B. Handwriting Numeral Data Set						
30	10	12	0.5	0.5	0.5	7
			0.3	0.5	0.5	9
			0.1	0.5	0.5	10
150	10	5	0.5	0.5	0.5	5
			0.3	0.5	0.5	5
			0.1	0.5	0.5	5

6.3 Concluding Remarks

A hybrid approach to automatic design of BP networks has been proposed. By combining the progressive training algorithm proposed for network growth and the node pruning algorithm proposed for network pruning, an effective hybrid network construction algorithm has been formed to fully self-determine the required network size. We have taken use of the guaranteed convergence property of progressive training to construct a solution network with finite number of hidden nodes first, and the four categories of excessive nodes involved are then removed afterwards according to the user's pruning specification. Satisfactory results were obtained. In fact, the growth and pruning processes in the present approach are exclusive with each other, and hence they could be replaced by the other more effective algorithms, if any.

7 CONCLUSIONS

To conclude, we summarize the contributions of the present work, note a few limitations, and suggest potential areas of further work.

7.1 Contributions

We have solved the network design problem of BP artificial neural networks by devising an automatic network construction algorithm composed of a network growth algorithm called progressive training and a network pruning algorithm called node pruning. The main contributions of this work are as follows :

(i) A Reliable Way to Network Growth

The work of progressive training conducted is the first approach to constructing BP networks by training with an expanding data set, which enables the network to grow gradually and reliably as the complexity of the current training data increases. The algorithm has been guaranteed to converge for any desired continuous one-to-one or many-to-one input-output mapping which is not achievable by existing network growth algorithms. Thus, it could be applied to any function approximation task, say image compression and time-series prediction. In addition to its faithfulness in constructing finite networks, progressive training is free from local minimum problems, an extraordinary property that is still missing from other ANN learning paradigms. In the worst case, the algorithm will find a network with the number of hidden nodes equal to the theoretical upper bound (see Theorem 1).

(ii) A Metric of "excessiveness" for Network Pruning

In previous network pruning methodologies, unnecessary nodes are removed from the network according to some sorts of system error measurements. It is unlikely that the internal roleplay of these nodes is transparent through such kind of metrics and hence we do not know whether a node could be pruned with preserved performance. The concept of "excessiveness" advocated here has been tried to close this gap and oversized networks could

be pruned with an explicit guideline. Thus, the pruned network is expected to be optimized with both the size and performance. Besides, this work has provided an analysis of the characteristics of hidden nodes in BP networks which helps to understand more on this class of ANNs.

(iii) A Hybrid Approach

By network growth, design effort on the network size is not required since it simply starts with a small network, but there still exists unnecessary nodes and links in the constructed network. By network pruning, an optimal or nearly optimal network would be obtained in the expense of predetermining the initial network size which requires *a prior* knowledge about the task such as its complexity. Taking the advantages of both approaches, a hybrid approach which employs the proposed network growth algorithm to construct a reasonably oversized network for the proposed node pruning algorithm to attain an appropriate network has been presented. This opens a gate to automatic design of ANNs.

7.2 Limitations and Suggestions for Further Research

(i) Progressive Training

It has been observed that much of the progressive training's learning time has been wasted by training previously learned patterns, and therefore it is slow. For this, improvements could be obtained if we train more for the new pattern and relatively train less for the old data and this has left for further research. Besides, the speed can also be improved by using any of the fast versions of BP algorithm (see, for example, [19-22]). Another fruitful area of further work is to cater for incremental learning, in which new information is added to the already-constructed network. Since new training patterns may not satisfy the largest magnitude requirement of progressive training (see Lemma 1 and Theorem 1), each of which may require more than one hidden node to handle. Preliminary conception is that at most two hidden nodes are needed for each new available training pattern since two parallel hyperplanes can be used to cluster it if the two hyperplanes are designed not to include any other

trained patterns also. We have analysed the sensitivity of a parameter of progressive training called STEADYE which specifies when a new node should be added. If this parameter is set too large, there is a waste of training time. If it is set too small, the non-convergence decision will sometimes be wrongly taken, and the size of the network may grow unnecessarily. A better theoretical understanding of the convergence criterion of the BP algorithm would be very helpful in order to optimally control this parameter.

(ii) Node Pruning

The node pruning algorithm was developed from a pattern classification point of view with its applications also restricted to the pattern classification tasks because the concept of category had been employed by the algorithm to detect excessive nodes. Hence, generalization of the algorithm to be applicable to the other tasks will be the focus of further work. Artificial neural network is also characterized by its high degree of fault-tolerance. Is it really tolerable if the damaged nodes happen to be the solution nodes rather than the excessive nodes ? That will be a very interesting area for further research.

(iii) Multi-hidden-layer Networks

Although Hornik et al. [26] and Funahashi [27] have proved the tremendous ability of one-hidden-layer networks, it has been suggested that networks with more layers, and fewer nodes in each layer, may generalize better than "shallow" networks with many nodes in each layer [40]. However, networks with many layers are far harder to train than networks having one hidden layer only. Therefore, extension of this work to multi-hidden-layer networks is suggested for further research with emphasis on improving the generalization performance and reducing the training time.

REFERENCES

[1] J.J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proc. Natl. Acad. Sci. USA*, vol.79, pp.2554-2558, April 1982.

[2] D.H. Ackley, G.E. Hinton, and T.J. Sejnowski, "A Learning Algorithm for Boltzmann Machines," *Cognitive Science*, vol.9, pp.147-169, 1985.

[3] T. Kohonen, *Self-Organization and Associative Memory*. New York:Springer-Verlag, 2nd ed., 1988.

[4] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing, Vol.I and II.*, D.E. Rumelhart and J.L. McClelland, eds., Cambridge, MA:MIT Press, 1986.

[5] D.E. Rumelhart and J.L. McClelland, eds. *Parallel Distributed Processing, Vol.I and II*. Cambridge, MA:MIT Press, 1986.

[6] B. Kosko, "Bidirectional Associative Memories," *IEEE Trans. System, Man & Cyber.*, vol.18, no.1, pp.49-60, 1988.

[7] G.A. Carpenter and S. Grossberg, "A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine," *Computer Vision, Graphics, and Image Processing*, vol.37, pp.54-115, 1983.

[8] G.A. Carpenter and S. Grossberg, "ART 2 : Self-Organization of Stable Category Recognition Codes for Analog Output Patterns," *Applied Optics*, vol.26, pp.4919-4930, Dec 1987.

[9] G.A. Carpenter and S. Grossberg, "ART 3 Hierarchical Search : Chemical Transmitters in Self-Organizing Pattern Recognition Architectures," in *Proc. Int. Joint Conf. on Neural Networks*, vol.2, pp.30-33, Wash., DC, Jan. 1990.

[10] O.K. Ersoy and D. Hong, "Parallel, Self-Organizing, Hierarchical Neural Networks," *IEEE Trans. Neural Networks*, vol.1, no.2, pp.167-178, June 1990.

[11] E. Yair and A. Gersho, "The Boltzmann Perceptron Network : A Soft Classifier," *Neural Networks*, vol.3, pp.203-221, 1990.

[12] A. Waibel, T. Hanazawa, G.E. Hinton, K. Shikano, and K. Lang, "Phoneme Recognition Using Time-delay Neural Network," *IEEE Trans. Acoust., Speech, Signal Proc.*, vol.37, no.3, pp.328-339, March 1989.

[13] D.G. Elliman and R.N. Banks, "Shift Invariant Neural Net for Machine Vision," *IEE Proceedings, Part I*, vol.137, no.3, pp.183-187, June 1990.

[14] S. Chen, G.J. Gibson, C.F.N. Cowan, and P.M. Grant, "Adaptive Equalization of Finite Non-Linear Channels Using Multilayer Perceptrons," *Signal Processing*, vol.20, no.2, pp.107-119, 1990.

[15] K.S. Narendra and K. Parthasarathy, "Identification and Control of Dynamical Systems Using Neural Networks," *IEEE Trans. Neural Networks*, vol.1, no.1, pp.4-27, Mar. 1990.

[16] R.P. Gorman and T.J. Sejnowski, "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets," *Neural Networks*, vol.1, pp.75-89, 1988.

[17] A. Khotanazad and J.H. Lu, "Classification of Invariant Image Representations Using a Neural Network," *IEEE Trans. Acoust., Speech, Signal Proc.*, vol.38, no.6, pp.1028-1038, June 1990.

[18] J.A. Benediktsson, P.H. Swain, and O.K. Ersoy, "Neural Network Approaches Versus Statistical Methods in Classification of Multisource Remote Sensing Data," *IEEE Trans. Geoscience and Remote Sensing*, vol.28, no.4, July 1990.

[19] R.A. Jacobs, "Increased Rates of Convergence Through Learning Rate Adaptation," *Neural Networks*, vol.1, pp.295-307, 1988.

[20] L.W. Chan and F. Fallside, "An Adaptive Training Algorithm for Back-Propagation Networks," *Computer Speech and Language*, vol.2, pp.205-218, 1987.

[21] S. Becker and Y. Le Cun, "Improving the Convergence ofBack-Propagation Learning with Second Order Methods," in *Proc. of the 1988 Connectionist Models Summer School*, D. Touretzky, G. Hinton, and T. Sejnowski, eds., pp.29-37, San Mateo, CA:Morgan Kauffman, 1989.

[22] T. Tollenaere, "SuperSAB : Fast Adaptive Back Propagation with Good Scaling Properties," *Neural Networks*, vol.3, pp. 561-573, 1990.

[23] A. Lapedes and R. Farber, "Nonlinear Signal Processing Using Neural Networks; Prediction and System Modeling," TR LA-UR-87_2662, 1987.

[24] G.W. Cottrell and P. Munro, "Principal Component Analysis of Images Via Back-Propagation", in *Proc. of Visual Communications and Image Processing '88*, SPIE Vol.1001, pp.1070-1077, 1988.

[25] A. Rajavelu, M.T. Musavi, and M.V. Shirvaikar, "A Neural Network Approach to Character Recognition," *Neural Networks*, vol.2, pp.387-393, 1989.

[26] K. Hornik, M. Stinchcombe, and H. White, "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks*, vol.2, pp.359-366, 1989.

[27] K. Funahashi, "On the Approximate Realization of Continuous Mappings by Neural Networks," *Neural Networks*, vol.2, pp.183-192, 1989.

[28] N. Baba, "A New Approach for Finding the Global Minimum of Error Function of Neural Networks," *Neural Networks*, vol.2, pp.367-373,1989.

[29] M.L. Brady, R Raghavan, and J. Slawny, "Back-Propagation Fails to Separate Where Perceptrons Succeed," *IEEE Trans. on Circuits and Systems*, vol.36, no.5, May 1989.

[30] R. Hecht-Nielsen, *Neurocomputing*. Addison-Wesley, 1990.

[31] Y. Le Cun, "Generalization and Network Design Strategies," in *Connectionism in Perspective*, R. Pfeiger, Z. Schreter, F. Fogelman and L. Steels, eds, Elsevier, Zurich, Switzerland, 1989.

[32] E.B. Baum and D. Haussler, "What Size Net Gives Valid Generalization ?," *Neural Computation*, vol.1, no.1, pp.151-160, 1989.

[33] J. Denker, D. Schwartz, B.Wittner, S.A. Solla, R. Howard, L. Jackel, and J. Hopfield, "Large Automatic Learning, Rule Extraction and Generalization," *Complex Systems*, no.1, pp.877-922, 1987.

[34] Y. Chauvin, "A Back-Propagation Algorithm with Optimal Use of Hidden Units," in *Advances in Neural Information Processing Systems 1*, D.S. Touretzky, ed., Morgan Kauffman, pp.519-526, 1989.

[35] S.J. Hanson and L.Y. Pratt, "Comparing Biases for Minimal Network Construction with Back-Propagation," in *Advances in Neural Information Processing Systems 1*, D.S. Touretzky, ed., Morgan Kauffman, pp.177-185, 1989.

[36] M.C. Mozer and P. Smolensky, "Skeletonization : A Technique for Trimming the Fat from a Network via Relevance Assessment," in *Advances in Neural Information Processing 1*, D.S. Touretzky, Ed., Morgan Kaufmann, pp.107-115, 1989.

[37] K.G. Beauchamp, *Walsh Functions and Their Applications*. London:Academic Press, pp.171-178, 1975.

[38] Y. Le Cun, J.S. Denker, and S.A. Solla, "Optimal Brain Damage," in *Advances in Neural Information Processing 2*, D.S. Touretzky, Ed., Morgan Kaufmann, pp.598-605, 1990.

[39] M. Mezard and J.P. Nadal, "Learning in Feedforward Layered Networks : The Tiling Algorithm," *J. Physics A : Math. Gen.*, vol.22, pp.2191-2203, 1989.

[40] D.E. Rumelhart, "Parallel Distributed Processing," *Plenary Session, IEEE Int. Conf. on Neural Networks*, San Diego, CA, 1988.

[41] M. Frean, "The Upstart Algorithm : A Method for Constructing and Training Feedforward Neural Networks," *Neural Computation*, vol.2, pp.198-209, 1990.

[42] S.E. Fahlman and C. Lebiere, "The Cascade-Correlation Learning Architecture," in *Advances in Neural Information Processing 2*, D.S. Touretzky, Ed., Morgan Kaufmann, pp.524-532, 1990.

[43] Y. Hirose, K. Yamashita, and S. Hijiya, "Back-Propagation Algorithm Which Varies the Number of Hidden Units," *Neural Networks*, vol.4, pp.61-66, 1991.

[44] N.J. Nilsson, *Learning Machine*. New York:McGraw-Hill, 1965.

[45] F. Rosenblatt, *Principles of Neurodynamics*. Washington DC:Spartan Books, 1961.

[46] P.J. Werbos, "Beyond Regression : New Tools for Prediction and Analysis in the Behavior Sciences," Doctoral Dissertation, Appl. Math., Harvard University, Nov. 1974.

[47] D.B. Parker, "Learning-logic," Technical Report TR-47, Center for Computational Res. in Economics and Management Sci., MIT, April 1985.

[48] A.E. Bryson and Y.C. Ho, *Applied Optimal Control.* [Revised Printing of the 1969 Edition], New York:Hemisphere Publishing, 1975.

[49] M. Minsky and S. Papert, *Perceptrons.* Cambridge MA:MIT Press, 1969.

[50] Y.H. Pao, *Adaptive Pattern Recognition and Neural Networks.* MA:Addison-Wesley, 1989.

[51] B. Widrow and S.D. Stearns, *Adaptive Signal Processing.* New Jersey:Prentice-Hall, 1985.

[52] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis.* New York:John Wiley & Sons, 1973.

[53] T. Lee and F.L. Chung, "A BP-Watchdog Process for Training Multilayer Perceptrons," in *Proc. Int. Conf. on Automation, Robotics, and Computer Vision '90,* Singapore, pp.260-264, Sept. 1990.

[54] J. Sklansky and G.N. Wassel, *Pattern Classifiers and Trainable Machines.* New York:Springer-Verlag, 1981.

[55] R.A. Fisher, "The Use of Multiple Measurements in Taxonomic Problems," *Ann. Eugenics*, vol.7, pp.179-188, 1936.

APPENDIX

A.1 A Handwriting Numeral Recognition Experiment : Feature Extraction Technique and Sampling Process

A handwriting numeral recognition experiment was conducted to evaluate the introduced algorithms in this thesis. The problem however is in no way a real world application but is sufficient to illustrate the effectiveness of the proposed algorithms. In this experiment, raw data were taken from three writers. In order to facilitate size and position invariant recognition, each input numeral was fitted into a box whose size and position were calculated from the maximum and minimum coordinates of the numeral. The raw data were then feature extracted using Walsh functions, as described in [25]. In this technique the numeral data are expanded in a set of orthogonal functions. The expansion coefficients form the features of the numeral. Before meaningful features are extracted, the numeral image undergoes a binarization stage based on a fixed threshold. The features extracted then depend on the intensity distribution of the binarized image. The general equation is of the form

$$L(x) = \sum C_n f_n(x) \quad \text{for } 0 \le x \le 1 \tag{A.1.1}$$

where $L(x)$ is the intensity distribution function which is expanded in a series of n known orthogonal functions $f_n(x)$. The set of coefficients C_n obtained from the above expansion characterizes the numeral data represented by the function $L(x)$. In this experiment, two intensity distribution functions $L_h(x)$ and $L_v(x)$ for the horizontal and vertical directions of the image were defined where x is the space variable in pixel length and was normalized to the interval $0 \le x \le 1$. By taking the Walsh transform of $L_h(x)$ and $L_v(x)$, the corresponding Walsh coefficients C_{hi} and C_{vi},

$$C_{hi} = \int_0^1 L_h(x) f_i(x) dx, \tag{A.1.2a}$$

$$C_{vi} = \int_0^1 L_v(x) f_i(x) dx \tag{A.1.2b}$$

where $f_i(x)$ is the ith Walsh Function [37], were used as the numeral's features. Hence each numeral resulted in a unique set of expansion coefficients, making it possible to categorize the numerals based on these coefficients. This was due to the disparity in the shapes of different numerals, which result in varying intensity distributions. Four Walsh functions were used for each distribution function and therefore the dimension of each numeral sample was eight. A total of 300 samples which composed of 30 samples from each of the ten numerals were collected for this experiment.

A.2 Determining the distance $d = \delta^2/2r$ in Lemma 1

In this appendix we describe the derivation of the distance $d = \delta^2/2r$ illustrated in Lemma 1. Figure A.2.1 depicts the explanations in the proof of Lemma 1 with hyperspheres $C(\vec{x}_{l+1}, \delta)$ and $C(o, r)$ intersect at points $I_1 \& I_2$ of I. We want to find the distance d. By looking at triangle with vertices o , I_1 & x_{l+1},

$$\cos\theta = \frac{r-d}{r} = 1 - \frac{d}{r} \qquad (A.2.1)$$

and

$$\cos\psi = \frac{d}{\delta} \qquad (A.2.2)$$

Since $\theta = 180° - 2\psi$, equation (A.2.1) becomes

$$-\cos 2\psi = 1 - \frac{d}{r} \qquad (A.2.3)$$

and hence

$$\frac{d}{r} = 1 + \cos 2\psi = 2\cos^2\psi \qquad (A.2.4)$$

Substituting equation (A.2.2) into equation (A.2.4), we have

$$\frac{d}{r}=\frac{2d^2}{\delta^2}$$

$$d=\frac{\delta^2}{2r}. \qquad (A.2.5)$$

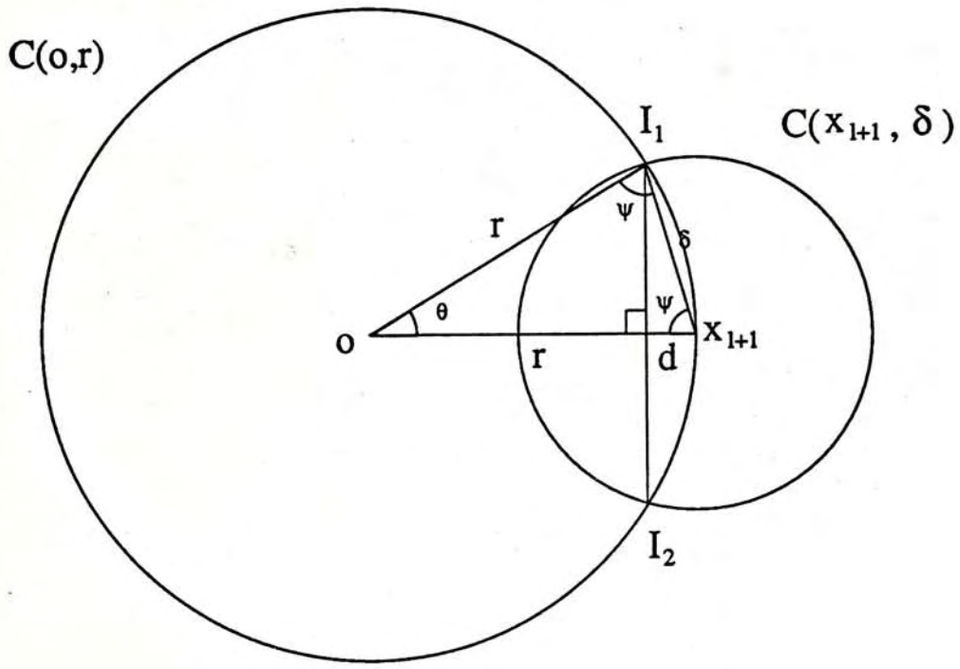

Figure A.2.1 Schematic illustration of Lemma 1's proof