ON THE TRAINING OF FEEDFORWARD 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 Hau-san Wong

June, 1993

UL

Thesis
QA
76.87
W644
1993

ABSTRACT

For the training of feedforward neural networks, the Back-Propagation (BP) algorithm has almost become the de-facto standard of training algorithms. However, the BP algorithm suffers from three serious defects, namely that the training speed is in general slow, that the training process may encounter local minima, and that the hidden layer size of the network has to be determined arbitrarily. The last problem is in particular serious as an inaccurate determination of the hidden layer size would indirectly lead to the other two problems.

Recently, a new training approach known as the dynamic node creation approach has been derived to counteract this latter problem of indeterminate architecture. The method employed by this approach is to start with a minimal network and subsequently add hidden nodes to the network when the need arises. In this way, an inadequate estimation of the hidden layer size can easily be compensated. However, this new approach of training has created other problems: for example, these kinds of algorithms either do not possess a convergence proof or its convergence depends on some artificial parameter initialization method for the new hidden node which in turn depends on a few patterns with large errors such that the action of the new node can offset their errors. This approach will in turn lead to the memorization of noisy patterns. Moreover, these dynamic node creation algorithms often start with a single-node network which leads to a long training time, and which in turn leads to poor generalization capability.

In this thesis, a deterministic dynamic node creation training algorithm is described. The distinctive feature of this algorithm lies in its deterministic new hidden node initialization scheme which depends on the whole training set rather than a single training pattern. As a result the memorization of a single noisy training pattern is discouraged. Moreover, the current algorithm is guaranteed to converge to a finite resulting network.

Two enhancements to this algorithm are also derived: the generalization measure monitoring scheme is derived to select the most suitable moment during training for the addition of a new hidden node to the network, thus preventing an overly long waiting time before the addition of a new node which leads to a deterioration in the generalization capability of the network. The derivation of the initial hidden layer size estimation scheme allows the determination of an initial hidden layer size for the dynamic node creation scheme instead of using an initial single node network, thus leading to less node additions in the dynamic node creation process which translates in turn to a shorter training time.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my supervisor, Dr. T. Lee , who has provided me with ceaseless inspirations and support during the course of my research, and who has helped me to discover the real pleasure of research.

I would also grateful to my colleagues in the Image Processing Laboratory, especially K.W. Cheung, Y.M. Tham, S.W. Chiu, W.K. Lai and H.K. Kwan for the many stimulating exchanges.

Finally, I would like to offer my deepest thanks to my family for their unfailing support throughout the course of my research.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Learning versus Explicit Programming

The von Neumann computer, the most prominent intelligent device today besides the human brain, essentially operates by following a set of verbal instructions known as programs supplied by us human beings. Through this operation mode the von Neumann computer excels at almost every task ranging from number crunching to database management. It gradually seems that almost every task conceivable by human beings can be translated into programs which is then carried out promptly by the sequential computer.

It was when the sequential computer found its way into wider areas of applications (especially in the field of pattern recognition) that the limitations of the above approach starts being felt: a simple object recognition task which we human beings take for granted, would quickly overwhelm the von Neumann computer due to the requirement for exact specification of the object before the possibility of recognition by the sequential computer. An entity such as a human face which could be recognized at a glance by us would have to be, according to the above scheme, broken down into individual feature and supplied to the computer. But in what way should we break up the human face to obtain those features (since obviously there is an infinite way of doing so) and how many of these features are to be supplied to the computer (since we do not know how many of these features are required to uniquely determine a human face)? Even if we can successfully supply these primary features to the computer, the computer still could not be qualified as an effective face recognizer due to the following fact: a human being can still recognize a face even if the face is somewhat distorted or if the person has aged. For the sequential computer, a complete re-specification of the features are required due to the variations in many of the original features. After the re-specification, there is no guarantee that the sequential computer would still recognize the face. In other words, the sequential computer pattern recognizer has zero fault tolerance and zero generalization. To continually recognize this face, we must program the sequential computer such that the two sets of features are associated with the single face. As can be visualized from the above example, the rule for face recognition would become overwhelmingly complicated if more examples of subtle variations of the same face are to be presented to the computer.

We can obtain some hints in solving this problem by questioning our own ways of recognizing such complicated patterns: the answer is that we simply do not know how we perform the task! In other words, no matter by what methods we are using in recognizing patterns, they simply cannot be expressed in verbal forms. As a result we cannot communicate this method to the sequential computer since it only accepts instructions in the form of verbal commands, and the only way to render a machine capable of performing such complicated pattern recognition tasks is to allow it to learn the rule by itself, since no supply of rules from the outside world is now possible. In fact, such a machine would mimic the human brain more closely as the human brain also learns its own rules.

1.2 Artificial Neural Networks

One would expect that such a learning machine would possess structures drastically different from the conventional sequential computer. In fact, the artificial neural network, an example of such a learning machine, derive its structure directly from the arrangement of neurons in the human brain: the neurons in the human brain are interconnected in a dense fashion to each other. Therefore, the signals in one neuron can simultaneously broadcast to many other neurons. In other words, the human brain operates in a highly parallel fashion, and this mode of operation may be responsible for the human brain's capability of recognizing complicated patterns. Researchers [50] therefore adopt this structure as their model of learning machine in the hope that the pattern recognition capability of the human brain can be directly grafted onto their machines.

The multilayer feedforward network proposed by Rumelhart et. al [49] is an example of such an artificial neural net in which the network is composed of layers of artificial neurons or nodes. The network consists of an input layer of neuron from which the network accepts external information and an output layer which emits the processed information. Between the two layers there exists multiple layers of neurons known as the hidden layers which perform nonlinear transformation on the incoming information. Each neuron in each hidden layer implements a nonlinear function and is connected to every neurons in the layer below or above through network weights which are real multiplicative factors. In general, the multilayer feedforward neural network is considered a mapping device which performs a mapping from R^n (n being the no. of neurons in the input layer) to R^m (m being the no. of neurons in the output layer). The functional shape of the mapping is defined by the hidden layers and the network weights. In this way, the topology of the learning machine in the form of the feedforward multilayer perceptrons are completely defined.

1.3 Learning in ANN

With the above learning machine, the process of learning can simply be defined as the self-adaptation of all the connection weights in the network such that the network implements a particular mapping specified by the user. In general, the mapping can be conveniently specified by extracting sample points from the desired mapping and presents them to the network as training examples. The collection of all these training samples is known as the training set. With the training set defined, we still have to find an effective way of modifying all the weights in the network such that the network implements the mapping at least at those training samples.

A training method can be developed for the feedforward neural network according to the following train of thoughts: for every possible combination of weights in the network, a set of network output is defined for every training sample. In general, the network output do not correspond to the desired output according to the specified mapping, and as a result a mean square error E is defined which is simply equal to the L^2 distance between the desired mapping and the network implemented by the network. Viewed in another perspective, a function E(**w**) is defined with the all network weights concatenated into a vector **w** as the function domain and the mean square error E as the function range. This function is normally known as the error surface [20]. In general, the shape of the errcr surface is unknown and varies with the mapping specified by the user.

At first sight it seems that the above alternative viewpoint in the form of error surface do not offer any further insight in developing a training method for the multilayer feedforward network. However, in an intellectual leap, Rumelhart et. al [49] proposed that one should descent on the error surface from an arbitrary initial state on the surface. Through this successive descent process the weights in the network will change in such a way that the system error E will decrease. Eventually the error E will be low enough such that the network implemented by the network will approximate sufficiently the desired mapping. In general, the simplest descent procedure on a continuous surface is the steepest gradient descent process which requires only the evaluation of first order derivatives.

Therefore, the evaluation of the partial derivative of E with respect to all the network weights is the prerequisite for the successful implementation of the learning process. The evaluation of this quantity has been successful performed for a primitive version of the multilayer feedforward networks which is known as the perception [48]. However, for many years, no attempts have been made to generalize the perceptron though it is well known that the perceptron cannot solve some very simple problems such as Exclusive OR as pointed out by Minsky and Papert [36], due to the misconception that the evaluation of the partial derivative of E with respect to all the weights in the network, if the network is anything more complicated than the perceptron, would be extremely difficult. Therefore the contributions of Rumelhart et. al [49] are such that they not only point out the feasibility of the generalization of the perceptron into the multilayer perceptron but in fact even derive the explicit expression for the partial derivative of E with respect to every weights in the network. The achievement of this task is made possible through the adoption of the sigmoid nonlinearity which possesses continuous derivative at every place instead of the step function used in perceptrons. In this way , the calculation of the various derivatives with respect to the network weights is made possible since the derivative expression usually involves the derivative of the nonlinear function in the node. Moreover, they also discovered that the partial derivative of E with respect to those weights connected to the hidden nodes in the lower layer can be expressed as a function of the partial derivative with respect to those weights connected to the upper layer. In other words, the errors are back-propagated from the upper network layers to the lower network layers, hence the name back-propagation algorithm adopted by Rumelhart et. al.

1.4 Problems of Learning in BP Networks

The BP network derived by Rumelhart et. al [49] has since then applied to many problems including image compression [38], time series modelling [58], handwritten character recognition [47] and many other pattern recognition problems. Though the BP algorithm has met with some success in almost every problems it is applied to, its various problems are also becoming more and more apparent as the algorithms is continually applied to new problems. In general, these problems can be categorised into the following three groups:

(1) Local Minimum problem: In principle, the BP algorithm is merely a gradient descent algorithm which simply seek out those regions with near zero derivative, while ignoring whether those regions are local minimum or global minimum. Many researchers had claimed that this problem is not important as through experimentation they have discovered that the probability of occurrences of local minima is low. However, Hecht-Nielson [20] had discovered through simulation studies that true local minima really exist. Moreover ,Gori et al [18] also proved that true local minima do exist in some very simple training sets. Therefore, the problem of local minimum is not merely a fictitious postulation but in fact poses a real threat to a network's eventual convergence.

(2) The Indeterminate Architecture Problem: Through simulation studies it is discovered that, in general, a neural network with more hidden layers and more nodes in each hidden layer can handle more complicated training sets. However, an analytical relationship between the complexity of a training set and the network size does not exist and the determination of the network size has to be based on trial and error. The most important theoretical contribution

till today towards this direction is the proof derived by Hornik et. al [23] that a neural network with only a single hidden layer can represent any arbitrary training set. As a result, the research work reported in this thesis involves only the single layer network as it is adequate to represent all possible training sets. Though equipped with this knowledge, we still have to estimate the number of nodes in the single hidden layer which is as much a delicate task: an underestimation of the hidden layer size will result in a network which is not capable of representing the training set, while an overestimation, far from being a bonus, actually causes a deterioration in the quality of training set representation and its subsequent generalization capability due to the over-abundance of parameters in the network which render the network capable of representing not only the general features of the underlying mapping but also any peculiarities in the particular training set chosen, including noise.

(3) The speed problem: The BP algorithm is in general found to be slow due to the requirement that the learning step size should be small such as to avoid oscillations of the network state during training. The lack of criterion in the choice of the learning step size also contributes towards this problem as researchers, in realizing the oscillation problem, adopt conservative learning steps for all of their problems when in fact the nature of some training sets would allow the adoption of larger training steps such that convergence can be achieved in a smaller number of training epochs (a training sweep through the whole training set). Moreover, there is also the ill-conditioned behaviour of the gradient descent algorithm on plateaus of the error surface: when the current state of the network is stuck on a plateau, a large weight adaptation would be required such that the network state can escape from its current position. However, due to the near zero gradient on the plateau the weight adaptation on the plateau would be extremely small, which is the opposite of what we require. This action of the algorithm would cause the current state to stuck at the plateau for a long time.

The best solution for the above problem would be to change the learning step size to suit the local curvature of the error surface. The simplest learning step control strategy is the momentum version of BP algorithm [49] which applies a judicious reinforcement or damping of the current learning step according to the training history. Notable among the attempts in directly controlling the learning step size according to the error surface profile include the works of Jacobs [26], Chan and Fallside [5], and Weir[59]. Significant improvements over the fixed learning rate version of BP were reported.

1.5 Dynamic Node Architecture for BP Networks

The three problems stated above are among the most serious problems of BP and are the main reasons which hinder the adoption of feedforward neural networks as a fully reliable engineering device. One may note that, among the three problems, the indeterminate architecture problem would pose a more serious threat to the eventual convergence of the network: the speed problem is at least partially tractable through the learning step size adaptation strategy due to the availability of information concerning the local curvature of the error surface. Though the existence of true local minima is confirmed through the works of Gori et al [18], the error surface may also contain many global minima due to the various permutations of the network weights among the different nodes and the chance of the current state reaching the global minimum would still be quite high. Moreover, we have no knowledge concerning the depth of the local minimum, and in many situations a local minimum with a sufficient depth would be almost as good as the global minimum itself.

However, the problem of indeterminate architecture is more intractable as we cannot simply pool resources into the network at our own expense: the over-abundance of parameters in the network would cause the over-fitting of the particular training set chosen to characterize the mapping, thus leading to a deterioration in the generalization performance of the network concerning the true underlying mapping of the training set. Moreover, when the number of hidden nodes in the network is inadequate for representing the training set, this indeterminate network will usually lead to an excessively long or even infinite training time without any apparent convergence of the network, and the resulting global minimum for the error surface would be so shallow that it would be almost as bad as a spurious local minimum. Thus the solution of the indeterminate architecture problem would ease the other two problems to a certain extent.

An apparent solution to the indeterminate architecture problem is the adoption of a dynamic node creation architecture for the neural network: if we do not know the number of hidden nodes required to adequately represent a training set, we can simply start with a single-node network, and subsequently add nodes to the hidden layer as the need arises. This seems a neat solution to the indeterminate architecture problem, but in fact the process of hidden node addition is an extremely delicate matter, since the initial values of the various parameters in the new node would have a large bearing on the subsequent convergence of the appended network: an improper initialization of the new hidden node will not only cause a stagnation of the error level of the appended network, it will actually disturb the information represented by the old network and cause an increase in the overall error level. This situation is vividly illustrated by the works of Hirose et. al [21]: they have applied the dynamic node creation strategy to the neural network in solving the XOR problem. This problem is famous for its complexity [50] but it is still fairly simply when compared to the higher order parity problems due to the small number of training patterns. This problem can readily be solved using a two-node network with the conventional BP algorithm. Hirose has attempted two strategies for node initialization: for the first case he initialized all the parameters with zero value such that there would be no disturbance to the information represented by the old network. In the second case he initialized all the parameters with random values. In applying this strategy to the XOR problem, it was discovered that on the average, as many as four nodes are added to the network before the error start to decrease, while on the average only two nodes are required for a conventional BP network to solve the problem. Therefore, the addition of a new hidden node to a network does not guarantee that the error will subsequently decrease, and the dynamic node creation process is not as straightforward as it first seems.

Moreover, a convergence proof would be indispensable for a dynamic node creation algorithm due to the unidirectional growth of the hidden layer, unless a node pruning process is also incorporated into the algorithm. Otherwise, there would be the possibility that the node addition process would go on forever without solving the problem ,especially for those schemes in which no effective node initialization strategies are adopted. I n view of the above difficulties, several improved dynamic node creation algorithms are derived which attempt to satisfy the above two criteria: that both a systematic node initialization scheme and a convergence proof must exist. Most notable among these efforts are the derivation of the Tiling Algorithm by Mezard et. al [34] and the Upstart Algorithm by Frean [13]. For the tiling algorithm, new hidden layers are successively defined for the old network and after which hidden node after hidden node is added to the new layer. Mezard et. al proved that the overall error of the network will decrease by a finite amount after the addition of a new layer, and therefore the network will eventually converge to the desired solution when an adequate number of hidden layers are generated by the algorithm. For the Upstart algorithm, the convergence of the network is facilitated by the inclusion of the so-called "daughter" units: when a unit in the network gives an erroneous response, a "daughter" unit is attached to the erroneous unit and correct its response. A proof of convergence is also given for the Upstart algorithm by its discoverers.

Although the above attempts represent a large step forward from the practice of arbitrarily initializing and adding nodes to a neural network, their algorithms are far from complete: the above two algorithms can only cater for binary inputs and outputs as opposed to arbitrary real inputs and outputs, as their convergence proofs rely heavily on the interpretation of the neural networks as a classification device and the hidden node as the hyperplane which separates two classes. This interpretation lends to a simplification of the formulation of the convergence proof, but the resulting algorithm cannot guarantee convergence when it is applied to real-valued mappings. In other words, the above algorithms are sacrificing the generality of their applications for an absolute guarantee on their eventual convergence.

Attempts have also been made to adopt the dynamic node creation procedure to real-valued mappings. The most notable example being the cascade-correlation algorithm derived by Fahlman et. al [12]. The approach attempts to adopt the correlation measure between the error vector of the old network and the outputs of the new hidden node as the new cost function for optimization. The algorithm is successfully applied to the parity problem and the 2-spiral problem, but no convergence proof has been presented by the two discoverers.

Recently, a new algorithm called the progressive training algorithm [9] has been derived. This algorithm involves only a single-layer network as opposed to the above algorithms in which multiple hidden layers are involved. In fact, such a node creation algorithm involving single-layer networks should be feasible in view of the proof by Hornik et. al [23] that a single layer network is adequate to represent arbitrary training sets. The name "progressive" is used as there is not only progressive growth in the hidden layer but also progressive growth in the training set: the initial training stage involves only a single training pattern, and patterns after pattern is added to the training set during the subsequent training stages until the whole training set is reconstructed. This extra growth process is required in order to guarantee the convergence of the algorithm: in fact the convergence proof of the algorithm is built around this extra growth process for the training set. The progressive training algorithm is the first dynamic node creation algorithm which is guaranteed to converge for both binary and real-valued training set. However, the algorithm is not without its problems, its problems mainly arise due to the extra requirement for the growth of the training set for its convergence, which would not be normal for ordinary BP networks in which the whole training set is utilized to train the network at any training stage, and its new hidden node initialization scheme which depend only on a single training pattern. These problems will be discussed further in Chapter 3. The most serious among these problems include the inability of the progressive training algorithm in catering for incremental learning.

1.6 Incremental Learning

It is typical for simulation studies in BP networks to adopt a training set with fixed size. However, it is not atypical for a training set to adopt a continuously increasing size. Examples of these kinds of training set include speech signals or music signals which usually arrive in continuous stream, time series modelling [58], power load forecasting [43], and channel equalization [7] where the training environment would gradually vary throughout the active operation phase of the neural network. In other words an online network training process has to be incorporated into the network such that it can continually adapt its weights to accommodate the new training patterns while retaining its memory for the old training patterns. As more and more problems of the above nature are considered solvable by the artificial neural network, researchers are also trying to derive an incremental learning scheme for the network. For example, Park et. al [43] derived a novel gradient scheme which can applied to a fully-trained network to adapt all its weights such that the resulting network can accommodate one extra training pattern on the assumption that the statistics of the incoming stream of training patterns vary slowly. However, the approach is restricted to only a fixed-size network: if training patterns are continually being added to the current training set, there would be a size threshold beyond which the capacity of the current network can no longer handle, unless we prepare to append extra hidden nodes to the network. In other words, a dynamic node creation strategy would be indispensable for a network which incorporates incremental learning. Therefore, if we generalize our concept of training set to include this class of dynamically expanding training set, we would realize that the dynamic node creation algorithm would be the corresponding generalization of our current idea of a neural network training algorithm.

1.7 Research Objective and Thesis Organization

In view of the dominant position of the indeterminate architecture problem among the three problems of BP, and the important role which would be played by the dynamic node creation strategy in the incremental learning process, we would focus on the learning behaviour of BP networks under the dynamic node creation environment. In particular we would like to develop a universal dynamic node creation strategy which is as problem-free as possible: the definition of problem-free includes the inclusion of a convergence proof in the overall scheme (unlike the cascade-correlation algorithm), the catering for real-valued training patterns as well as binary-valued pattern (unlike the Tiling Algorithm and the Upstart algorithm), the non-requirement for a specific ordering or partitioning of the training, and the provision for incremental learning (unlike the progressive training algorithm). In addition, it has been seen through the works of Hirose et. al [21] that training a fixed-size network and training a network which is dynamically expanding are very different tasks. It would be useful if we can compare the quality of the mapping implemented by neural networks trained using both methods. A reliable indicator concerning this aspect is the generalization capability of the trained neural network on a testing data set which is generated independently from the training set. We have brought up this issue as we expect that a neural network trained using a dynamic node creation algorithm would in general exhibit a poorer generalization rate due to the uneven training received by each hidden node in the network as compared to the hidden nodes in a conventional BP network in which uniform training is received. Measures are to be derived to counteract this problem such that the generalization capability of the dynamic network should at least approximate that of the conventional BP network.

A brief introduction to the contents of each chapter in this thesis is now appropriate: in chapter 2 a brief synopsis of the historical development of the perceptron and the multilayer feedforward perceptron will be given. The Back-Propagation algorithm will be derived and the notations concerning the network weights, and the various node inputs and outputs will be introduced. In Chapter 3 we would review the various attempts made by researchers in understanding how the neural network represents a training set through its weights, which culminates in the discovery of the T-vector approach which would be useful in our subsequent work. The progressive training algorithm is also introduced in this chapter and its various advantages and shortcomings are discussed. In Chapter 4, we introduce the first step in our research work which culminates in the derivation of the growth algorithm: the discovery of this algorithm allows the application of the dynamic node creation strategy to an unordered training set as opposed to the progressive training algorithm where an ordered training set according to the Euclidean distance is required to ensure convergence, and which in turn requires the availability of all training patterns prior to training and precluding any possibility for incremental learning. Therefore the growth algorithm opens the first step in the implementation of a dynamic node creation algorithm which is capable of performing incremental learning. In Chapter 5, the T-vector approach in characterizing the various parameters of the neural network is introduced. This alternative approach of characterisation is instrumental in the formulation of the deterministic dynamic node creation algorithm introduced in Chapter 6: the need for this deterministic dynamic node creation algorithm, which allows the initialization of the new hidden node by multiple training patterns, preclude any possibility of the memorization of noisy training patterns which would happen under the progressive training algorithm and the growth algorithm. The deterministic algorithm represents a generalization of the above two algorithms and acts as our prototype for the universal dynamic node creation algorithm. In Chapter 7, a new node addition criterion is derived which is based on the generalization measure derived by Drucker et. al [11]: the improved criterion aims at alleviating the ill-conditioned representation inherent in dynamic node creation algorithms due to uneven training, such that the generalization measure of the dynamic network can approach that of the conventional BP network. Finally, in Chapter 8, attempts are made to estimate the initial network size prior to the start of the hidden layer expansion process such as to alleviate the uneven training problem which is manifested in its extreme form in a dynamic network built up from a single node architecture.

2 THE FEEDFORWARD MULTILAYER NEURAL NETWORK

2.1 The Perceptron

To begin any discussion on the feedforward multi-layer neural network, one must first discuss its famous predecessor, the perceptron. It was invented by psychologist Frank Rosenblatt [48]. It consists of a series of multipliers feeding into a summation device which produces a corresponding output from the two processes of multiplication and addition. This device is further improved by Widrow et al [61] into the Adaline which stands for the ADAptive LINear Element , in which a bias input is present beside the series of multipliers and in addition a hard limiter is present in the output which limits the output of the device to +1 to -1. Since these two devices are nearly identical we would concentrate on discussing the Adaline which is of particular importance to classification problems. The adaline is depicted in fig 2.1.

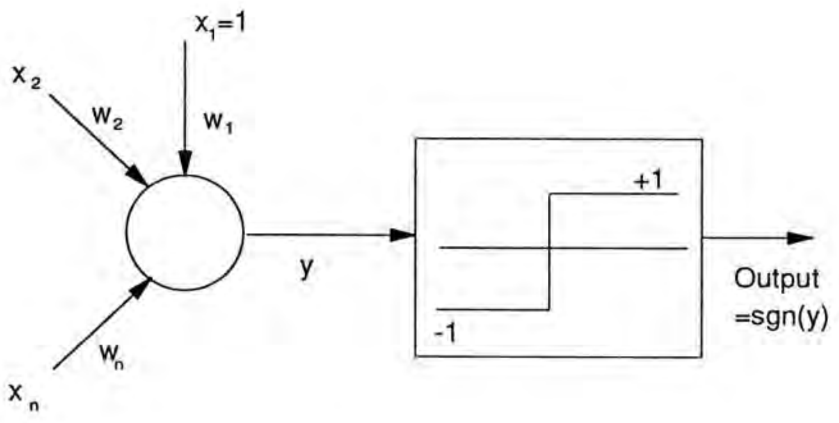

Fig 2.1 The Adaline

This device is particularly suitable for dealing with simple classification problem as the binary output of the device can be translated into whether the current training pattern belongs to class 1 or class 2 for a two-class problems. In general, for different classification problems which are represented by different training sets, the network weights w_i, i=1 to n would be different for the correct classification of all the training patterns, and these weights are normally obtained using a gradient descent procedure known as the delta rule. Suppose that the training set consists of p training patterns. At the presentation of each pattern, the delta rule attempts to decrease the classification error at that training pattern by changing all the network weights in the steepest descent direction on the error surface. In mathematical terms, each weight is to be changed according to Eq (2.1), in accordance with the steepest descent principle when the t-th pattern is presented to the network

$$w_i(t+1)=w_i(t)-\eta\frac{\partial E_t}{\partial w_i} \tag{2.1}$$

where

$$E_t=(d(t)-y(t))^2=(d(t)-\sum_{i=1}^{n} w_i x_i(t))^2 \qquad (2)(2.2)$$

The term d(t) represents the desired output for a particular training pattern. The partial derivative in Eq (2.1) can be conveniently calculated from Eq (2.2) as

$$\begin{aligned}\frac{\partial E_t}{\partial w_i}&=-2(d(t)-\sum_{i=1}^{n} w_i x_i(t))x_i(t)\\&=-2(d(t)-y(t))x_i(t)\end{aligned} \tag{2.3}$$

Since all the quantities in Eq (2.3) are readily available from the training pattern and the output of the Adaline, the training procedure can be promptly carried out. The parameter η in Eq (2.1) is called the learning step size and controls the speed of learning. The process continues until the classification error on the training set becomes zero. The action of the adaline on the training set can be depicted pictorially in Fig 2.2.

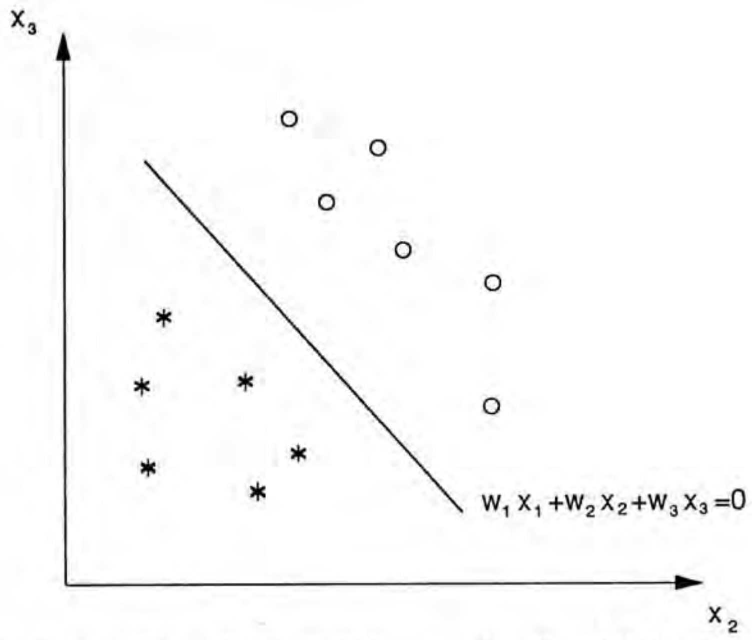

Fig 2.2 The separation of the training set by the Adaline

It is seen that the action of the input summation device is equivalent to the production of a hyperplane which partitions the input pattern space into two half-spaces. In one of the half spaces, the output of the adaline would be 1, while in the other half-spaces the output would be -1. The action of the delta rule is to change the orientation of the hyperplane in such a way as to minimize the classification error on the training set. The most important characteristic of the Adaline is that, unlike the design of traditional engineering device in which extensive analytical techniques are involved, the Adaline can instead learn its own parameters from the training set. Thus it possesses an ability which remotely resembles one of our abilities which distinguish us from machines, namely the ability of learning. The creation of such a device at that time has led to much controversy as it reminds people of science fiction stories where intelligent machines can in turn control the human race. Unrealistic expectations and exaggerate claims has no doubt lead to these controversies. However, in the next section, we would realize that the Adaline is in fact a very limited device in terms of the types of problems it can cater for.

2.2 The Generalization of the Perceptron

In 1969, a book appeared that had almost sounded the death knell for the whole field of neural networks. The book was titled Perceptrons and was written by Minsky and Papert [36]. They had shown in their book that the perceptron cannot in fact solve some very simple problems. They illustrated their viewpoints by showing that the perceptron cannot solve the Exclusive-OR problem. This problem is illustrated pictorially in Fig 2.3.

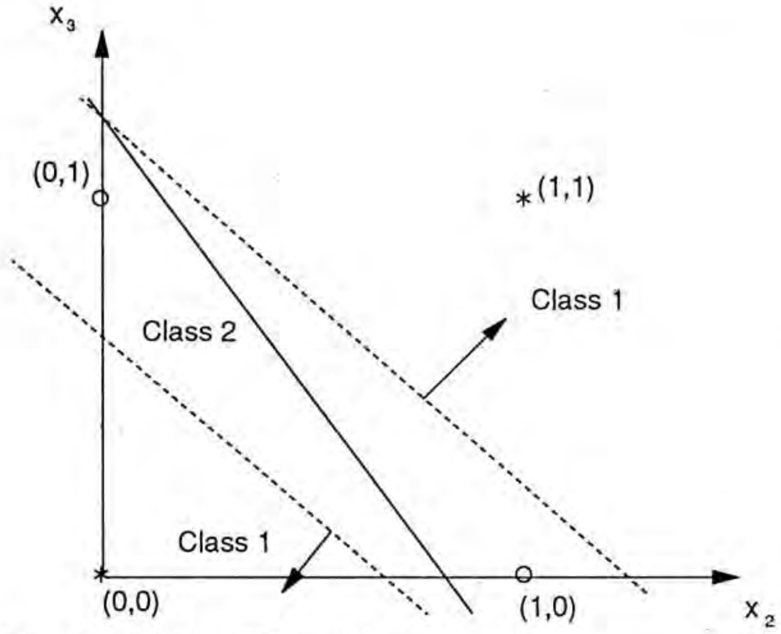

Fig 2.3 The Exclusive-OR problem

For this XOR problem, the training patterns (0,0), (1,1) belong to one of the classes, and the patterns (0,1) and (1,0) belong to the other class. It is immediately apparent that no matter how we change the orientation of the hyperplane, we cannot properly classify the XOR training set by the Adaline. In fact, this is only one of the many problems which the Adaline cannot solve. On the contrary, the class of problems which the Adaline is able to solve belongs to a very restricted class, which is called the class of linearly separable problems. Therefore, there is the need to extend the capability of the Adaline.

We could take our clue in how to generalize the Adaline from the above XOR problem: from Fig 2.3, it is obvious that if we can implement two hyperplanes which correspond to the class boundaries of the XOR problem, the problem can be readily solved. This implies the requirement of at least two Adalines. Moreover, the two Adalines should be configured such that two separate nonlinearities are required instead of using only one nonlinearity for two summation devices, as the summation of two hyperplanes is still a single hyperplane which is equivalent to the action of a single Adaline. Besides, one more nonlinearity is required which acts as the single output node for the whole network and accepts its input from the output nonlinearities of the two Adalines. In this way, we have extend the former simple Adaline into a multi-element, multi-layer network which has the potential to cater for complex problems.

However, the extension of the architecture for the Adaline has posed another problems: in the derivation of the delta rule, we are relying on the linearity of the summation device to calculate the partial derivative of E_t with respect to the network weights. For the extended device, since the network error occurs at the single output element of the entire network, we must somehow relate this error to the network weights of the two Adalines. However, any such relationship must involve the derivative of the individual nonlinearity of each Adaline. Since the sgn(x) nonlinearity is not differentiable at its transition, an effective training strategy does not exist for this extended network.

2.3 The Multi-Layer Feedforword Network

In 1985, Rumelhart et. al has, in their famous book [50], solved all the problems of the generalization of the perceptron in a single stroke. The result of this generalization is the emergence of the multi-layer feedforward network, which can in theory possess a large number of network layers with each layer containing multiple network nodes. Most importantly, Rumelhart et. al has derived an algorithm for training this massive network, which allows the neural network to be applied to a very wide class of problems, and which in turn leads to the revival of the field of neural network. Since this generalized learning algorithm involves the back propagation of the output errors from the uppermost layer of the network to the lowest layer of the network in order to determine the derivative of the error with respect to all the weights in the network, this algorithm is thus called the Back Propagation (BP) Algorithm.

Before introducing the solution proposed by Rumelhart et. al in solving this network extension problem, we would first introduce the architecture of this extended network which is depicted in fig 2.4.

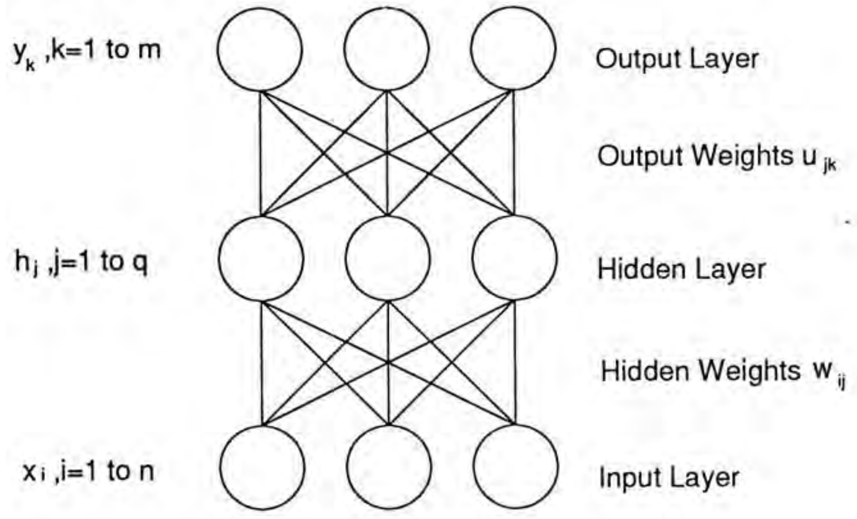

Fig 2.4 The feedforward multilayer neural network

From Fig 2.4 it is seen that the network consists of three types of layers, namely the input layer which contains n input nodes, the hidden layer which contains q hidden nodes, and the output layer which consists of m output nodes. In general, we would use the index i to identify the input nodes, the index j to identify the hidden nodes, and the index k to identify the output nodes. In principle, this kind of network can contain multiple hidden layers between the input layer and the output layer, with the connections between them resembling those between the hidden layer and the input or output layer. Each hidden node performs a nonlinear transformation on its inputs as opposed to the Adaline in which a linear node is used. Since our research work mainly involves the single hidden layer network, we would present here the single-layer version of this class of networks. The network weights between the input layer and the hidden layer is known as the hidden weight and is denoted by w_{ij}. The network weights between the output larger and the hidden layer is known as the output weights and is denoted by u_{jk}.

This network is usually considered as implementing a mapping from $\mathbf{R}^n$ to $\mathbf{R}^m$ in which the totality of all input node values are considered as vectors in $\mathbf{R}^n$ and the totality of all output node values are considered as vectors in $\mathbf{R}^m$. From here on, we would identify each node output value by two parameters. For example, each input node value is described by $x_i(t)$, where i identifies the position of the node in the input layer and the parameter t, which ranges from 1 to p, identifies which pattern in the training set is being presented to the network. Similarly, the hidden node output is identified by $h_j(t)$, and the output of the output node is given by $y_k(t)$. In addition, the desired output of the network for the various training patterns is given by $d_k(t)$ which corresponds to $y_k(t)$. In this thesis, we would designate the node $x_1(t)$ as the bias node, i.e: $x_1(t)=1$ for all t, and $h_1(t)$ as the bias hidden node.

Throughout this thesis, we would adopt two principal ways of naming the parameters of the network, namely the spatial vector or S-vector approach and the temporal vector or T-vector approach. These two kinds of naming convention would be described in detail in Chapter 5, and we would like to introduce our way of symbolizing these two kinds of vectors. The Spatial vector or S-vector approach corresponds to our usual way of concatenating the node outputs of each layer into a vector. As a result, we could suppress the subscript i, j or k when identifying these S-vectors, while using the bold-face type to indicate these S-vectors. For example, the input S-vectors are $\mathbf{x}(t)$, the hidden S-vectors are $\mathbf{h}(t)$ and the output S-vectors are $\mathbf{y}(t)$. On the other hand, the temporal vector format is an alternative way of concatenating the parameter of a neural network. In short, this approach concatenates the whole history of a single node through a sweep of the training set into a single vector. As a result, the parameter t is suppressed. The input T-vectors are $\mathbf{x}_i$, the hidden T-vectors are $\mathbf{h}_j$ and the output T-vectors are $\mathbf{y}_k$.

With the introduction of the various terminologies, we would now proceed to introduce how Rumelhart et. al solved the problem training such a massive network. The answer is that he adopted an alternative nonlinear function for the hidden nodes which is depicted in Fig 2.5.

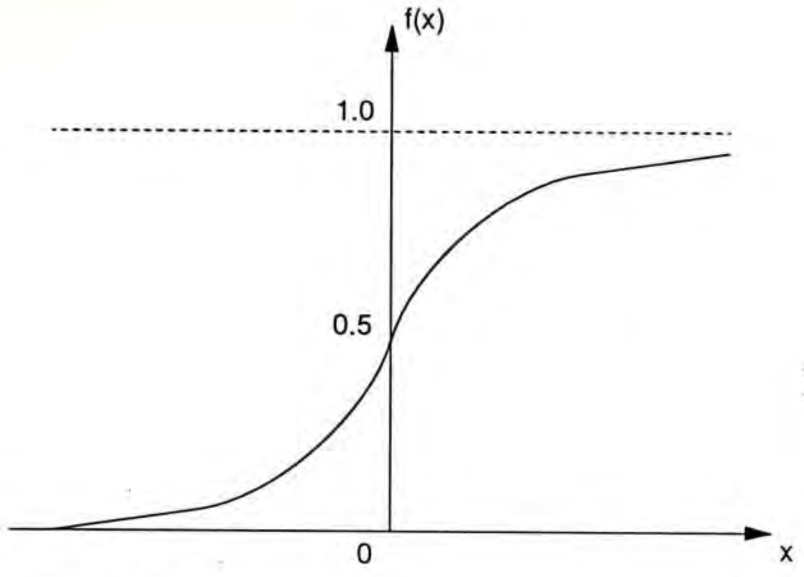

Fig 2.5 The sigmoid function

This function is known as the sigmoid function and is defined by the following relationship

$$f(x)=\frac{1}{1+e^{-sx}} \tag{2.4}$$

The factor s is a sensitivity factor for the function. When s becomes large, the sigmoid function would approach the step function. Through the definition of the sigmoid function, the derivation of the training equation for this multilayer network becomes possible, as the sigmoid function is differentiable at every point of the function and thus the derivative of the output error with respect to all the weights in the network can be readily evaluated. We would first summarize the action of the network. Since this action is independent of training patterns, we would suppress the parameter t: For the hidden layer:

$$\begin{aligned} a_j &= \sum_{i=1}^{n} w_{ij} x_i \\ h_j &= f(a_j) \end{aligned} \tag{2.5}$$

Where a_j as defined above is called the activation of the j-th hidden node. For the output layer,

$$b_k = \sum_{j=1}^{q} u_{jk} h_j \tag{2.6}$$

$$y_k = f(b_k)$$

Where b_k is called the activation of the output node. In this way the output of the network can be calculated from the above set of equations.

Defining the error function of the network as

$$E = \frac{1}{2} \sum_{t=1}^{p} \sum_{k=1}^{m} (d_k(t) - y_k(t))^2 \tag{2.7}$$

We would now present the Back-propagation equation for the adaptation of the hidden weights and the output weights. The derivation of these equations would be given in the Appendix. For the output weights u_{jk}, the equation is

$$\begin{aligned} u_{jk}(n+1) &= u_{jk}(n) - \eta \frac{\partial E}{\partial u_{jk}} \\ &= u_{jk}(n) + \eta \sum_{t=1}^{p} \delta_k^o(t) h_j(t) \end{aligned}$$

where

$$\delta_k^o(t) = y_k(t)(1 - y_k(t))(d_k(t) - y_k(t)) \tag{2.8}$$

While for the hidden weights w_{ij}, the equation is

$$\begin{aligned} w_{ij}(n+1) &= w_{ij}(n) - \eta \frac{\partial E}{\partial w_{ij}} \\ &= w_{ij}(n) + \eta \sum_{t=1}^{p} \delta_j(t) x_i(t) \end{aligned}$$

where

$$\delta_j(t) = h_j(t)(1 - h_j(t)) \sum_{k=1}^{m} u_{jk} \delta_k^o(t) \tag{2.9}$$

The factor η is the usual learning step-size which would hereafter be referred to as the network adaptation gain. The index n is the training epoch indicator where an epoch refers to a single presentation of all the training patterns to the network and the subsequent adaptation of the various weights.

Practically, an additional term is appended to the weight adaptation equation in order to speed up the training process as shown in the following expressions:

$$\begin{aligned} u_{jk}(n+1) &= u_{jk}(n) - \eta \frac{\partial E}{\partial u_{jk}} + \alpha(u_{jk}(n) - u_{jk}(n-1)) \\ w_{ij}(n+1) &= w_{ij}(n) - \eta \frac{\partial E}{\partial w_{ij}} + \alpha(w_{ij}(n) - w_{ij}(n-1)) \end{aligned} \tag{2.10}$$

From Eq (2.10), it is seen that these modified equations attempt to add a portion of the previous adaptation to the current adaptation equation and is thus maintaining the "momentum" of travel along the error surface. Due to this reason, the factor α is named the momentum factor and its value is usually between 0 and 1.

At this point the BP training algorithm has been fully presented. In the Chapter 3 we would review the various modifications to this algorithm such that its various problems as mentioned in Chapter 1 could be partially alleviated.

3 SOLUTIONS TO THE BP LEARNING PROBLEMS

3.1 Introduction

In searching for an effective solution to the various problems of BP, researchers have opened up almost every frontiers of attack imaginable: some of them set their aims in solving the three problem of BP learning: namely the speed problem, the local minimum problem and the indeterminate architecture problem. On the other hand, some researchers attempt to probe into the hidden layers of the network in order to decipher what is going on behind the hidden representation of the neural network, with the hope that once they have understood the hidden representation of the network for a certain training set, they can apply the reverse process of synthesizing the representation from the training set. In other words, they can then build up from scratch a neural network with solely the information of the training set without any further iterative learning procedure, therefore signifying the discovery of the ultimate deterministic learning algorithms for neural network which is the dream of every neural network researcher. Their attempts, however, are met with limited success due to the fact that, in order to build up a neural network without any iterative procedure, one cannot avoid dealing with the nonlinearity present in every hidden nodes of the network. Since the development of nonlinear analysis techniques lag much behind that of linear analysis techniques, researchers have no tools to deal with the nonlinearities effectively, and therefore must rely more or less on iterative procedures as part of the strategy in their overall training scheme. For example, the deterministic training algorithm described in Chapter 6 of this thesis only involves the deterministic estimation of the initial state of the network: the network must be guided towards its final state by means of iterative learning procedures. However, these attempts in understanding the internal representation of the network are not as futile as they seem, as we can, equipped with these new found knowledge, find ways of initializing the network in a way which is compatible with this knowledge instead of in a random way. The network would then start in a much more favourable position than the conventional arbitrary position a network is required to take up in previous training attempts:

in the terminology of the error surface and weight space, if a really excellent initialization scheme is derived for the neural network, the current state of the network could be placed just at the brim of the bowl-shaped global minimum such that the slightest iterative procedure would send the current state straight into the global minimum. If this ideal initialization scheme can be realized, it would not be of much difference from a truly deterministic scheme except for a few epochs of iterative learning cycles. However, the constraints for developing this initialization scheme would be much relaxed since we only require an approximate initial state for the network. The algorithm in Chapter 6 was also developed with this aim in mind and all endeavour described in this chapter concerning the hidden representation model should, at least for the present moment, seen in this light due to the extreme difficulties for an exact analysis of the hidden nonlinearities.

The first section in this chapter will trace the various endeavour in developing a model for the hidden representation of the neural network: the preliminary efforts in the direction attempt to understand the neural network in terms of spatial vectors or S-vectors: this term is developed to contrast the temporal vectors or T-vectors which is an alternative interpretation of the various node outputs of the network and the difference between these two terms will be clarified in Chapter 5: however, a single example here would serve to illustrate their differences: an input spatial vector would be what we usually call a training pattern; thus the concept of S-vector would conform to our usual interpretations of the various parameters of the neural network. For example, hidden S-vectors would be the transformed input patterns in the hidden space and an output S-vectors would be the output values of the neural network at the various output nodes concatenated into a vector. On the contrary, for a particular node of the network, we can concatenate the whole history of a node's activity into a vector, i.e. if the training set consists of p patterns, then the corresponding node output for each of the training pattern for a particular node can be concatenated into a vector called the temporal vector or T-vector since each component of the T-vector is related to each other temporally. In summary, the early researchers attempted to investigate the distribution of the hidden spatial vectors in the hidden space for a particular training set and to decide under what conditions would a hidden S-vector distribution considered valid for an adequate description of the training set. For example, for a training set which is not linearly separable and for a neural network which consists of a single output and a single hidden layer only, the input S-vectors should be transformed by the hidden layer in such a way that the resulting hidden S-vectors would be linearly separable in the hidden space, such that the hidden space can be properly partitioned by a certain hyperplane characterized by the output weights of the single output node. One can immediately realize the difficulties presented by the S-vector approach from the above example, as there are infinite ways of realizing a hidden representation which is linearly separable, and it is not at all clear which representation we should adopt for the current training set. In other words the hidden weights for the network cannot be uniquely chosen for a particular training set, and in any way we do not have any pre-conception about the number of dimensions in which the hidden space is to be embedded. Moreover, for a neural network which possesses multiple output nodes, the hidden representation should be designed such that it should be simultaneously linearly separable in multiple direction by hyperplane which are represented by the output weights of the various output nodes, which is an extremely difficult task since in the first place we do not know the orientations of these various hyperplane, not to say designing a corresponding hidden representation which is simultaneously linearly separable by them. The next section would trace these various efforts in understanding the internal workings of the neural network using this S-vector approach.

Realizing the futility in relying on the S-vector method for a realizable model for the hidden representation, researchers have recently turned to the T-vector or temporal vector method for an alternative representation of the network parameters. It was found out that the resulting model for the hidden representation is much simplified, as it involves only the relationship between the input space **X**, the space spanned by the input T-vectors the hidden space **H** which is the space spanned by the hidden T-vectors, and the inverse desired output space $\mathbf{D}^{-1}$ which is the space spanned by the inverse desired output T-vectors. These relationships will be further illustrated in Chapter 5. Through this new approach the problem of arranging the training patterns in the training set into a viable representation has simplified to the arrangement of a few T-vectors,into their proper positions, since the number of nodes in each layer of the network is usually much smaller than the number of training patterns, and the previous rather intractable criterion of linear separability has been converted to the more tractable criterion of distance measurement between a certain T-vector and its projection on another space. The development of the T-vector approach is still at a rudimentary stage and the application is restricted to the tuning of the relative positions between the hidden space **H** and the inverse desired output space $\mathbf{D}^{-1}$, which is equivalent to the searching for the optimal output weights u_{jk} for the various output nodes by means of linear least square optimization methods. The relationship between the input space **X** and hidden space **H** is still largely unknown due to the presence of the sigmoid nonlinearity between the two spaces which prevent the direct application of linear algebra to the above scenario, and in the deterministic algorithm described in Chapter 6, an attempt is made to relate these two spaces through a linear approximation on the sigmoid function which allows a direct contact between the two spaces. The various early efforts in building an internal representation model of the neural network using the T-vector approach would be traced in Chapter 5. In addition, the various terminologies in the above discussion will be clarified further in Chapter 5 which in addition includes the graphical depiction of the relationship between the various spaces.

The second subsection in this chapter would focus mainly on dynamic node creation algorithms. In fact, the establishment of a viable hidden representation model is intimately related to dynamic node creation algorithms as a valid hidden node representation model should ultimately include directions on how to select the dimension of the hidden space **H**, and until recently the most effective practical method in performing this task lies in these node addition algorithms. Besides, these classes of algorithm are important in their own rights: it has been seen that the indeterminate architecture problem can be considered the most serious problem among the three problems of BP, and since the dynamic node creation algorithms are among the few methods which can effectively deal with its problem, the establishment of a truly universal dynamic node creation algorithm would serve to eradicate a large portion of the problems inherent in BP learning. Moreover, it has been mentioned that such a node addition scheme would be indispensable for a training algorithm which performs incremental learning, and as a result these hidden layer expansion procedures should be considered standard appendages to future training algorithms as neural networks become exposed to the ever widening circles of applications.

3.2 Attempts in the Establishment of a Viable Hidden Representation Model

Every investigations and formulation of theories should start with empirical studies of the actual environment, and there is no exception for the formulation of a viable hidden representation model for neural networks: researchers at first try to observe the hidden patterns of trained neural networks to try to understand what is going on behind the hidden layer. We would state once again that these early endeavour are chiefly based on the spatial or S-vector model because it is the conventional way of understanding and categorizing the various parameters of the networks. In observing the hidden representation of a network trained with the XOR training set, Pao [42] has discovered that a once linearly inseparable training set (the set is linearly inseparable due to the disjoint nature of one of the classes) is transformed into a linearly separable data set at the hidden layer such that it can easily be partitioned by the hyperplane represented as the output weights of the hidden node. He thus concluded that the hidden layer has the effect of arranging the relative positions of the class clusters such that they are more separable by a hyperplane. Webb and Lowe [57] went a step further and established that for a neural network which is trained to minimize the mean square error at the network output, the function of the hidden layer is to maximize a discriminant function which is defined by the inter-class covariances divided by the total covariances. In this way, the empirical observations of Pao was translated into concrete mathematical terms which state that in essence the hidden layer tries to maximize the inter-class distance and separates the classes to as large an extent as possible. However, this insight cannot be translated into a viable algorithm for synthesizing the hidden representation as the above condition, that of transforming the classes such that they are more separable is an extremely broad one which is very difficult to establish except for the usage of iterative procedure, in which case there would be no differences from directly applying BP to the network. As a result more careful observations of the hidden layer activities are necessary. Gorman and Sejnowski [19], in analyzing a neural network trained to classify sonar targets, established that contrary to the popular belief that a hidden unit plays the role of a feature extractor, it can in fact encode multiple features and even multiple strategies simultaneously. Through this approach the neural network can make more efficient use of the capacity of each hidden unit. In other words, the hidden layer may be trying to establish a model for the training set rather than performing the task of simple feature extraction. This conjecture is further confirmed when they discovered that the network is able to internally encode pattern variations that do not decompose simply into a set of feature dimensions. Moreover, they also found that the network would memorize less frequent training patterns by using a small number of hidden weights to encode these peculiarities. Thus the hidden layer may be performing functions which are more complicated then we have usually assumed. Michaels [35] went a step further by establishing through close observations and subsequent tabulations of all the hidden weights w_{ij} and the output weights u_{jk} that, though the values of these weights vary greatly from each training trial, the Network Linear Transform matrix $\mathbf{N}=\mathbf{UW}$, where $\mathbf{U}$ is the matrix containing the output weights and $\mathbf{W}$ is the matrix containing the hidden weights, is highly invariant, thus indicating the mutually dependent nature of the output weights and hidden weights. He also discovered that the hidden weights which exhibit complex and highly variable waveforms containing multiple positive and negative peaks, are in fact producing simple differencing operations on the pattern set. This characteristic also explains how the hidden units, as observed by the author, are able to reject features that are common to all input patterns. Based on this observation, the author has produced a model which can quite accurately mimic the performance of a trained BP network. However, the construction of his model is based on the parameters of a trained network and therefore this method cannot be adopted in building a network from scratch. Moreover, though we now know that the hidden weights perform differencing operations on the input patterns, we do not quite know which input patterns among the patterns the network would choose for differencing. As a result, although the above studies provide interesting insights on the internal representation of neural networks and elucidate the roles played by the various weights in the network, a clear guide towards a deterministic construction of the internal representation is still lacking.

Due to the above rather inconclusive search for a deterministic hidden representation construction scheme from the observation of actual hidden representation alone, researchers have resorted to their own conceptions of how the hidden unit should process the input patterns in designing a viable construction scheme for the neural network: An example of these efforts is the recent resurgence of interests in using the radial basis function [37,55] as the hidden nonlinearity instead of the sigmoid nonlinearity, due to the resulting rather intuitive interpretation of how a hidden unit performs its task especially for classification problems: since the function value of a RBF $g(\mathbf{x})$ is dependent upon only the distance between the "centre" of the function and $\mathbf{x}$, it can frequently be pictured as a "bump" in the domain of the training set. For a classification problem, if we select the "centres" of the RBF's in the hidden units of the network to be the centroid of the various classes and adjust the "width" of each RBF appropriately to include all patterns belonging to that class, an adequate model for the training set will emerge. Tsoi [55] has derived criteria for selecting the "centres" for a RBF network since we have often no a priori knowledge on the class distribution of the training set. On the other hand, we do not get much insight frorm this model in designing networks for functional approximation, since these tasks cannot be conveniently described in terms of classes. Moreover, sigmoidal functions possess several advantages over the use of RBF for the approximation of functions in high-dimensional spaces as described in [58]. Therefore, researchers try to synthesize new types of nonlinearities to capture the essential features of sigmoid and RBF nonlinearity such that the hidden node possesses the advantages of both types of nonlinearities. The approximation scheme derived by Girand et. al [17] belongs to this class. In their scheme they adopt a pair of sigmoidal units as their elementary building blocks in building up the neural network. The pair of hidden nodes is configured in such a way that it performs a differencing operation on the two sigmoid functions of the two nodes. The resulting combined nonlinearity of the two nodes resemble an inverted gully on the domain of the training set which extends infinitely in its longitudinal direction and confined to a local region in the orthogonal direction. The authors attempted to construct arbitrary mappings through these inverted gullies by considering them as plane waves in the Hilbert space, and the representation for the function is obtained through a Fourier expansion. The advantage of this approach lies in its mathematical tractability since the Fourier expansion method is a well-established technique. The disadvantage of this method lies in the special arrangement of the hidden nodes which require a even number of hidden nodes in every network under this scheme. Since functional approximation is possible for BP networks using an odd number of hidden nodes, the above approximation scheme do not represent a realistic scenario of the learning process. Daunicht [10] went a step further in advocating his own conception of the function of the hidden layers by proposing a multiple layer network architecture with custom-designed nonlinearity at each layer which performs a specific function. In summary, hidden layers of the network attempts to divide the domain of the training set into fine grids, and the function of the output layer is to fill in the functional value of the mapping to be approximated in each of these grids. In principle, more accurate approximation of the mapping can be obtained by dividing the domain of the training set into finer and finer grids. Learning is confined to the output layer where the functional value at the various grids are to be adjusted to fit the particular mapping to be approximated. The parameters of the hidden layers are fixed since they are performing the same function of dividing the training set domain into grids no matter what mappings are to be approximated. The advantage of this approach is that we are not required to determine the architecture of the hidden layers since they are essentially fixed, which can equally be its disadvantage since mapping with simpler structures cannot be represented by a smaller network and thus signifies a waste of resources.

Clearly realizing the difficulties of the above hidden representation modelling approaches which involves the spatial vector approach, researchers have turned to the temporal vector or T-vector approach to find an alternative perspective in order to probe into the inner workings of the neural network. These efforts will be summarized in Chapter 5 together with an introduction to this T-vector approach.

3.3 Dynamic Node Creation Algorithms

We would now proceed to the discussion of dynamic node creation algorithms: we would in particular discuss the progressive training algorithm [9] in detail since it is the first dynamic node algorithm which can cater for function approximation tasks and at the same time possess a convergence proof. The class of dynamic node algorithms has become more and more important in recent years due to the increasing complexity of the training set being exposed to the neural network which makes the determination of the size of the hidden layer a more and more elusive task. Moreover, the emergence of the concept of incremental learning through a special class of training set including time series modelling and power load forecasting has render the inclusion of dynamic node creation algorithms almost mandatory. This class of algorithms would also be indispensable for the modelling of the hidden representation in which it plays the role of dimensionality determination for the hidden space **H**, since there is until now no effective analytical technique for determining this dimension. The very early efforts towards this objective are directed towards binary training data set: in the tiling algorithm proposed by Mezard et. al [34], the architecture of the network is not fixed in advance and is generated by the algorithm itself. The algorithms add hidden layers, and units inside a layer, until the network converges. In each layer, the growth of the hidden unit is initiated by a so called master unit which ensures that the error of the network decreases strictly from one layer to the next. After that, ancillary units are added to the hidden layer in order to get "faithful" internal representations for the training data, the definition of "faithful" being that different training patterns should possess different internal hidden representation. Mezard was also able to prove that through this scheme, the number of misclassification of the data set decreases from one layer to the next by virtue of the master unit, and as a result the algorithm will eventually generate a network which possesses finite number of hidden layers and finite number of hidden nodes within each layer.

Unfortunately, since the convergence proof is based on the strict decrease of the number of misclassifications of the training data, the algorithm is applicable only to training set with binary outputs and this constitutes one of the main drawback of this algorithm. Frean [13] proposed an alternative method called the Upstart Algorithm in which a hierarchical structure of hidden nodes are suggested to build up a network which is smaller in size than that built up by the tiling algorithm. In summary, the network starts with a single node Z, and if the node is "wrongly ON" for some of the training patterns, meaning that the network emits a 1 where the desired output should be 0, a new unit called the "daughter" unit is appended to the previous "parent" unit which produces a strong inhibitory signal at the "wrongly ON" pattern. Similarly, if a pattern causes the node to be "wrongly OFF", a new "daughter" unit which produces a strong excitatory signal is appended to the "parent" unit. In a similar way, if the daughter unit makes any mistakes, new daughter units are appended to these units, and the previous daughter units become parent units. As a result, the overall classification error decreases at every addition of daughter units. Eventually, none of the terminal daughters units make any mistakes, which in turn implies that their parents do not make any mistakes, and their parents, and so on. Therefore the above process will eventually produce a network which is capable of classifying all the training patterns. The main drawback of this algorithm, however is the same as the tiling algorithm, it can only cater for training set with binary output since the convergence proof is based on the "wrongly ON" and "wrongly OFF" concept.

The first dynamic node creation algorithm which can cater for real-valued training data set is the cascade correlation algorithms proposed by Fahlman et. al [12]. In this algorithm, the network starts with no hidden unit, and then trained using the delta rule. If the network does not converge, a new hidden unit is added to the network. Before the addition, the hidden weights of the units are trained such that the correlation between the error of the old network and the output of the hidden unit is maximized. The unit is then added to the network with its hidden weights frozen, and the output weights of the network are then continuously trained to obtain a solution. A special feature of the node addition scheme is that a new hidden node not only receives inputs from the input node, but also from all the previous hidden units added to the network as well, thus the resulting network in fact possesses a pseudo-multi-layer architecture. Satisfactory performances have been reported by applying the algorithm to the parity problem and the 2-spiral problem. By the above scheme the network is able to cater for real-valued outputs since the correlation measure is also defined for real numbers. However, a convergence proof of the algorithm is lacking, and the growth of the network is in principle not restricted. Furthermore, it is not clear why Fahlman has adopted such a specialized architecture for his network instead of the usual single-layer BP network. Perhaps he has found through experiments that the freezing of all the hidden weights of the network will severely limit the degrees of freedom of the network, and this has to be compensated by the incorporation of high-order feature detectors in the network. Moreover, as pointed out by Michaels [35], the Network Linear Transform (NLT) which is defined by the multiplication of the output weight and hidden weight matrices are highly invariant among the various training runs for a single training set, though the individual hidden weights matrix and the output weight matrix are found to vary widely from trial to trial. This implies that the output weights and hidden weights may be highly dependent on one another, and the separate training of these two types of weights as adopted by the cascade correlation algorithm may not constitute an efficient training strategy.

Hirose [21] has attempted to adopt the dynamic node creation strategy to a conventional BP network with a single hidden layer. He has no particular node initialization strategy in mind and attempted to initialize the new node with random weights. However, it turned out that the addition of a new node does not automatically signify an immediate corresponding decrease in error. For example, when the above scheme is applied to the XOR problem which normally requires a two-node network, the overall error of the network does not start to decrease until the addition of the fourth node. In view of this phenomenon, Hirose has adopted a pruning procedure for the network in order to counteract the network growth rate. The experimental results also indicate the definite need for an initialization scheme for the new hidden node which may help in guiding the overall network to the global minimum.

The progressive training algorithm proposed by Chung et. al [9] has derived such a node initialization strategy for a single hidden layer BP network. Moreover, a convergence proof is given for the training scheme, such that the training will terminate with the generation of a finite-sized network. However, the convergence is achieved at the cost of adopting an artificial procedure for feeding the training patterns to the network: all the training patterns in the training set are first ordered according to their Euclidean distances, and the training starts with a single-node network and a single training pattern. If the network is unable to converge a new hidden node which is initialized according to a definite scheme is appended to the network which ensures the convergence of the network under the partial training set by the virtue of this very scheme. If the network is able to converge a new training pattern with the next largest Euclidean distance is appended to the training set. This process continues until all the training patterns are appended to the training set. The resulting neural network must be finite in size as the convergence of the network under the addition of one more training pattern require at most one more new hidden node by virtue of the node initialization scheme which will be introduced below:

Suppose that pattern **x**(t) is recently added to the training set and the network is unable to converge. Suppose also that at this stage the network contains Q hidden nodes. Thus the non-convergence signifies the requirement for the Q+1-th hidden node. The hidden weights and hidden bias are initialized according to the manner below:

$$\begin{aligned} w_{iQ} &= s x_i(t) \qquad\qquad 2 \le i \le n \\ w_{1Q} &= -s \sum_{i=2}^{n} x_i^2(t) \end{aligned} \qquad (3.1)$$

The above initialization scheme allows the hyperplane of the new hidden node to act as a barrier which separates the old training patterns from the new pattern as depicted in fig 3.1.

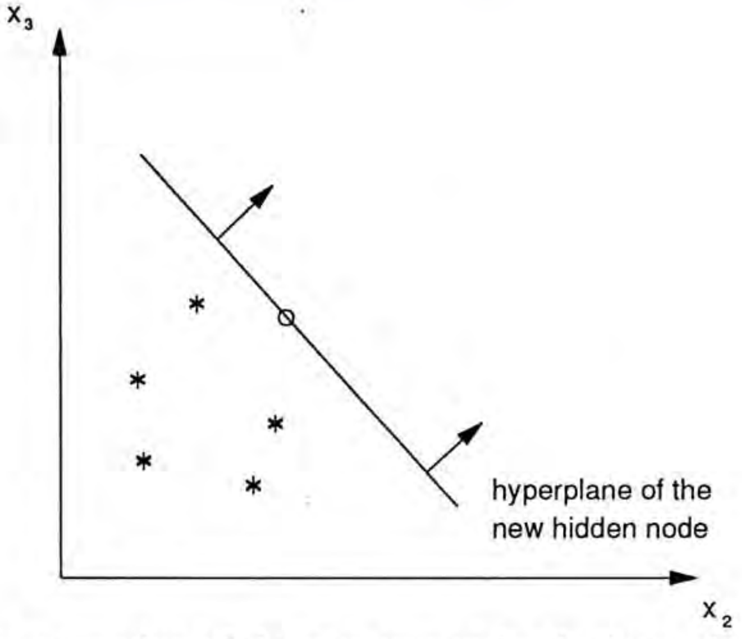

Fig 3.1 The separation of the new pattern by the hyperplane of the new node

The direction of the arrow indicates the increasing direction of the sigmoid surface of the new node. The factor s in Eq. (3.1) is a scale-up parameter which controls the slope of the sigmoid function. As the value of s increases, the sigmoid surface will more and more approach a step function, and the disturbance of the new node to the old patterns will approach zero. By virtue of the above node initialization scheme, the output of the new node will remain at 0.5. As a result, the height of the sigmoid surface can be scaled by the output weight of the new node in such a way that the error at the new training pattern can be exactly compensated when s is large enough. Practically, we start at s=1 and apply BP to the resulting network. Since the node output at the new training pattern is always 0.5, the output weights u_{Qk} of the new node is determined by

$$u_{Qk}=\frac{f^{-1}(d_k(t))-\sum_{j=1}^{Q-1} u_{jk}h_j(t)}{0.5} \tag{3.2}$$

Prior to applying BP, the current state of the network is recorded, and if the network does not converge after the BP process, the previous state of the network is restored and s is increased. In view of the above consideration the network will eventually converge when s is large enough.

The progressive training algorithm seems to be the ideal algorithm which is long being sought after, since it both possesses a convergence proof and can cater for both binary and real-valued training set. However, there are several problems inherent in the implementation of this algorithm:

(1) It is seen that the new hidden node is initialized by the information of a single training pattern only. There exists possibilities that the training set will be contaminated with noise. If incidentally a new hidden node is initialized with such a pattern, it will probably lead to the memorization of such a noisy pattern which is alien to the rest of the training patterns and in turn leads to the overall misrepresentation of the training set.

(2) Due to the artificial training sequence, training patterns with small Euclidean norms are exposed more often to the network than patterns with large Euclidean norms since they are present in the partial training set early in the training stages and they are not removed as training proceeds. Therefore, the error at those patterns with small norms are usually smaller due to the heavy training received, while patterns with large norms often invoke relatively large error since they receive less training. As a result the overall training process is biased towards those patterns nearer to the origin.

(3) It will be shown in Chapter 4 that the progressive training algorithm is not capable of performing the task of incremental learning unless excessive resources are allocated to the network, due to the requirement that the training set must be ordered according to the Euclidean distance prior to training. If a new pattern which is to be added to the training set does not satisfy the maximum norm criterion, then according to the above scheme, the resulting network is not guaranteed to converge even if a new node which is initialized according to the new pattern is added to the network.

(4) The algorithm starts with a single node network. As most neural networks which can cater for common training sets require multiple hidden nodes, a single node network

would be the most unlikely network which can adequately represent a practical training set. If the resulting network which approximates the target training set would be large, it would take a long time for the algorithm to build up the eventual network.

In view of the above problems, there exists a need for a new neural net training algorithm which can solve these problems. The dynamic node creation nature of the algorithm would be retained due to its many desirable features and in particular its indispensability in the implementation of incremental learning. This implies that an alternative node initialization scheme or even an alternative node addition strategy should be derived. The course of our research work would be mainly directed towards the realization of these two goals.

3.4 Concluding Remarks

In this Chapter we have reviewed past research works on the modelling of the hidden representation of a neural network. We have in addition described various attempts in deriving a dynamic node creation algorithm for the neural network. These two issues are placed side by side in this Chapter since they are actually intimately related: any valid model for the hidden representation of a neural network must include directions on how to select the dimension of the hidden space, and till today dynamic node creation serves as a viable and simple strategy for deciding the hidden layer size of the network. The implementation of the dynamic node creation strategy depends, in turn, on a better model of the hidden representation in order to design a more accurate new hidden node initialization scheme of the network. In particular, the progressive training scheme are described which represents an advanced stage of the development of the dynamic node creation technique since it possesses both the ability of catering for real-valued training set and a convergence proof, in which one condition or both is lacking in all of the previous node addition algorithms. Therefore, it seems that the progressive training scheme serve as a good starting point for our further course of research. In Chapter 4, we would first develop a dynamic node creation algorithm based on the overall node addition strategy of the progressive training algorithm but with the removal of the Euclidean distance ordering requirement of the training set such that the new algorithm would be suitable for operation in an incremental learning environment.

4 THE GROWTH ALGORITHM FOR NEURAL NETWORKS

4.1 Introduction

In this chapter we will develop a new dynamic node creation algorithm for neural network. In the previous chapters, we have introduced the cascade correlation algorithm [12] and the progressive training algorithm as examples of dynamic node creation algorithms. The emphasis of an effective hidden node initialization scheme would reduce the error invoked by the appended network by substantial amounts such that an adequate representation of the training set would be achieved within the smallest number of hidden node additions. On the contrary, an inefficient node addition scheme would cause the resulting error of the appended network to stay at the same level or even increase to such an extent that the original useful information embedded in the original network would be lost. Ignoring the optimality issue at the moment, the node initialization scheme should at the very least cause a finite error decrease to the appended network such that the algorithm will generate a final network with a finite hidden layer size. In other words a proof of convergence is essential to the admissibility of a dynamic node creation algorithm. The cascade correlation algorithm has not provided such a proof, and therefore the algorithm is in the above sense incomplete. On the other hand, the progressive training algorithm has provided such a convergence proof, and in principle the algorithm will terminate with a resulting network which contains at most p - 1 hidden nodes, with p being the number of training patterns. But the node initialization procedure which leads to this convergence hinges heavily on the following conditions:

(1) All the training patterns must first be ordered according to their Euclidean distance

(2) The training set must first be broken down into its constituents and then reassembled pattern by pattern according to their Euclidean distances until the complete training set is restored. At each instant the network only sees a partial training set.

The gradual reconstruction of the training set according to the Euclidean distances is important in ensuring that at each stage of training there is only one novel pattern which will be involved in substantial training, while the other patterns, which have already been well-learnt by the old network, undergo only slight fine-tuning in order to accommodate the novel pattern. At the same time, the ascending Euclidean norm condition ensures that the novel pattern possesses the greatest norm among all the patterns contained in the partial training set, and there exists a new hidden node such that whenever the hidden node is added to the network, the error at the novel pattern will immediately go to zero, and since the remaining patterns have already been well-trained in the old network, we can immediately obtain a satisfactory representation of the partial training set through the addition of the new node. The construction of the new hidden node which satisfies this criterion has been described in Chapter 3: we can simply align the principal axis of the sigmoid surface of the new node (the principal axis of the sigmoid surface refers to that direction on the surface in which the variation is greatest) with the position vector of the novel pattern and shift the surface along the principal axis in such a way that the output of the new node at the novel pattern would be 0.5. The portion of the surface which asymptotically approaches zero should be closer to the origin than the portion which asymptotically approaches one such that the remaining patterns of the training set is not greatly disturbed by the addition of the new hidden node. Convergence of the algorithm is ensured by allowing the sigmoid function to approach the step function such that the disturbance to the old patterns in the training set by the new node approaches zero while the output of the new node at the novel pattern remains at 0.5. As a result, we can control the network output to whatever value we wish simply by scaling the height of the sigmoid function in the form of scaling the output weight of the new hidden node. The portion of the sigmoid surface which asymptotically approaches 1 would have no effect on the partial training set as by the very definition of the current partial training set there would be no training patterns in this region. The above process is repeated until the whole training set is reconstructed.

Despite the various problems with this node initialization scheme mentioned in the previous chapter such as its artificial training sequence and the low convergence speed, it is quite obvious to see from the above discussion that the progressive training algorithm would have difficulty in accommodating new information. First of all, we have to order all the training patterns in the training set according to their Euclidean distances, and this implies that all the information to be loaded onto the network must be available prior to the commencement of training. If new information are later appended to the training set, we cannot simply insert the new pattern into the old training set according to its Euclidean distance with respect to the old training patterns, as we cannot create a corresponding new hidden node which elicits zero error at the new training pattern, as the new training pattern may not possess the greatest Euclidean norm among all the training patterns in the training set. If we forcefully apply the previous node initialization scheme to prepare a new hidden node for the new training pattern, we would end up with the situation depicted in fig 4.1.

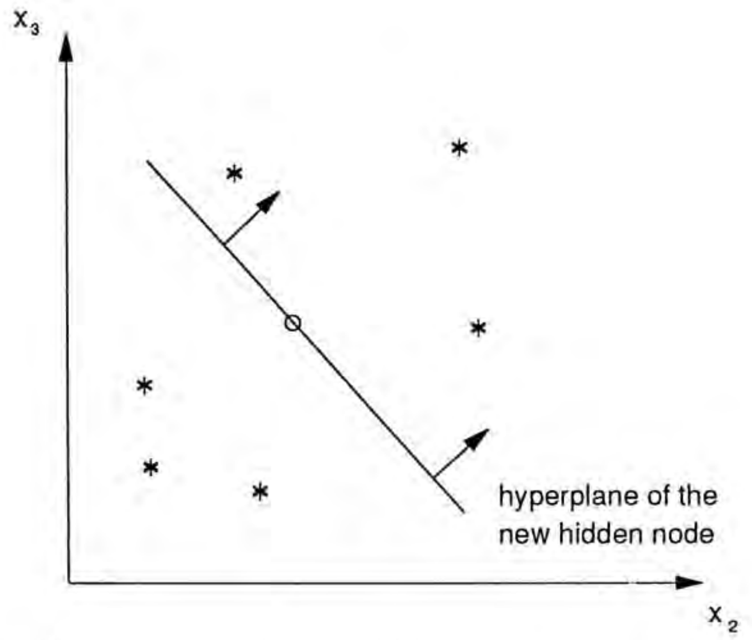

Fig 4.1 The disturbance of the new hidden node to the old training patterns

The arrow in Fig 4.1 refers to that portion of the sigmoid surface in which the function value asymptotically approaches 1. We can immediately see that even if we let the sigmoid function approach a step function, there would still exist a portion of the training set in which the disturbance of the new hidden node would be finite, which is exactly that portion in which the training patterns possess a greater Euclidean norms than the new training pattern. Although there is a possibility that the appended network would again settle down to a stable solution after the BP process, the previous convergence argument based on the new hidden node initialization scheme cannot go through, and the convergence of the network after the BP process may as well be attributed to sheer luck. Moreover, there is no guarantee that the above process will terminate with a finite network when more and more training pattern are appended to the training set. The author in [9], clearly anticipating this problem in the progressive training algorithm, attempts to surmount this difficulty by suggesting the utilization of a hidden node pair to bound up the pattern whenever a new training pattern not satisfying the maximum Euclidean norm criterion is appended to the training set. We can depict this solution in fig 4.2.

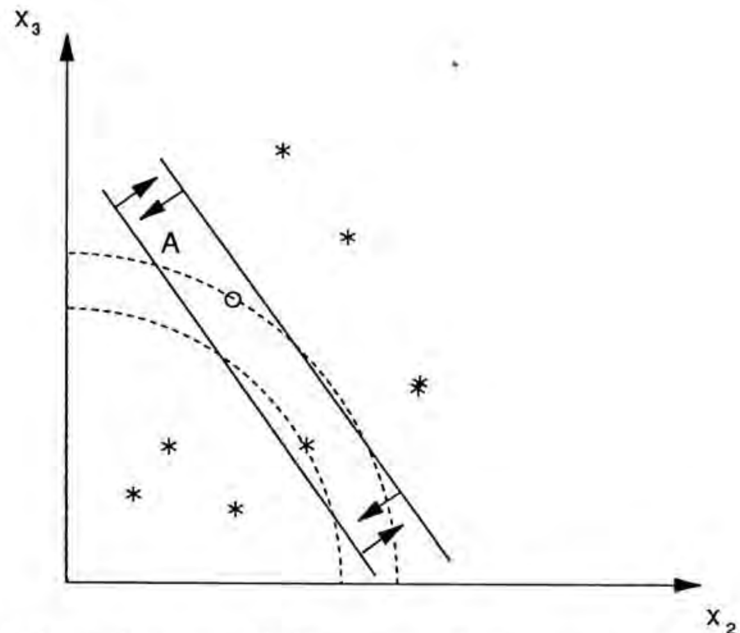

Fig 4.2 The bounding up of the new training pattern by a pair of hidden nodes

From fig 4.2 we can see that the double hidden node approach still cannot solve the problem described above, as these training patterns which fall within the region A would still be disturbed by the new hidden nodes. This is due to the fact that the sigmoid surface extends infinitely not only in its principal direction but also in the transverse direction as well. Though the double hidden node approach still cannot solve the disturbance problem, the region of disturbance has shrunk from a half plane to a narrow strip in the domain of the training set. As a result, the true solution to the above problem should be obvious by now: we need only two more hidden nodes in order to surround the new training pattern as depicted in fig 4.3.

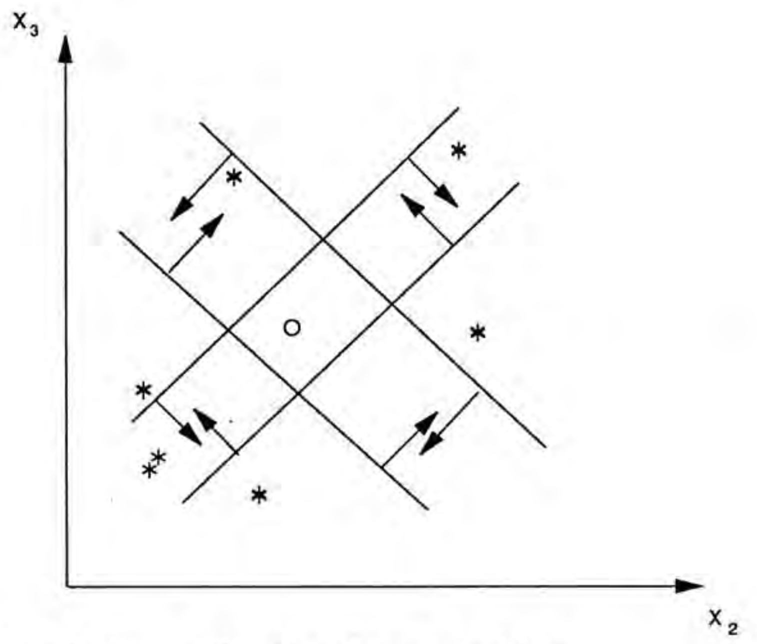

Fig 4.3 The solution to the disturbance problem

It can be seen that the four hyperplanes have truly solved the problem of disturbance by restricting the region of disturbance to a finite region in the vicinity of the new training pattern. However, the production of the four hyperplanes implies the addition of four hidden nodes to the network for only a single new training pattern. To further worsen the situation, if there is a stream of new training patterns to be appended to the network, the above node addition scheme, though ensuring a finite resulting network, will cause the hidden layer to expand to such an enormous size that its implementation in terms of hardware would be prohibitively difficult. Moreover the excessiveness of parameters in the network would most likely cause overfitting of the training set through the BP fine-tuning process. Therefore we can see that we are paying a high price by insisting on using the sigmoid function as the exclusive function for the hidden node under the above model.

4.2 The Radial Basis Function

The radial basis function has often been portrayed as an alternative function to the sigmoid function in the implementation of neural networks. The functional form for the radial basis function is given by

$$R(x)=g(\|x-c\|^2) \tag{4.1}$$

where **c** is the so-called centre of the radial basis function. It can immediately be seen that the functional value depends solely on the distance between the patterns and the centre of the function. The Gaussian function is most often used as the function g in the above expression

$$\begin{aligned} g(\|x-c\|^2) &= \exp\left(-\sum_{i=1}^{n}\frac{(x_i-c_i)^2}{2\sigma_i^2}\right) \\ &= \prod_{i=1}^{n}\exp\left(-\frac{(x_i-c_i)^2}{2\sigma_i^2}\right) \end{aligned} \tag{4.2}$$

where x_i and c_i are the components of **x** and **c** respectively. This function can be visualized as a local bump in the domain of the training set. The position of the bump can be changed by varying the centre **c**, and the width of the bump can be adjusted by varying the variance σ_i^2 of the various dimensions.

The advantage of using the RBF in the hidden node lies in its computational efficiency, as exemplified by Moody and Darken [37], since the adaptation of the network involves only the local updates of the relevant hidden node, as each hidden node only covers a finite region of the training set domain. In addition, the feasibility of using the RBF as a complete substitute for the sigmoidal nonlinearity is in the hidden units are firmly established by Park and Sandberg [44], who proved that the radial-basis function is a universal approximator. In general, the sigmoid function and the radial basis function have their own advantages and disadvantages and each cannot say to have outperformed the other. However, we can immediately see that a radial-basis function network is particularly suited to our problem under the above node creation environment. We have seen that at least four hidden units are required for enclosing a new training pattern which is to be appended to the old training set, and this represents an especially uneconomical distribution of resources (four hidden nodes versus one training pattern). With the radial basis function (assuming a Gaussian function for our RBF), the new training pattern can be enclosed by a single hidden node as depicted in fig 4.4.

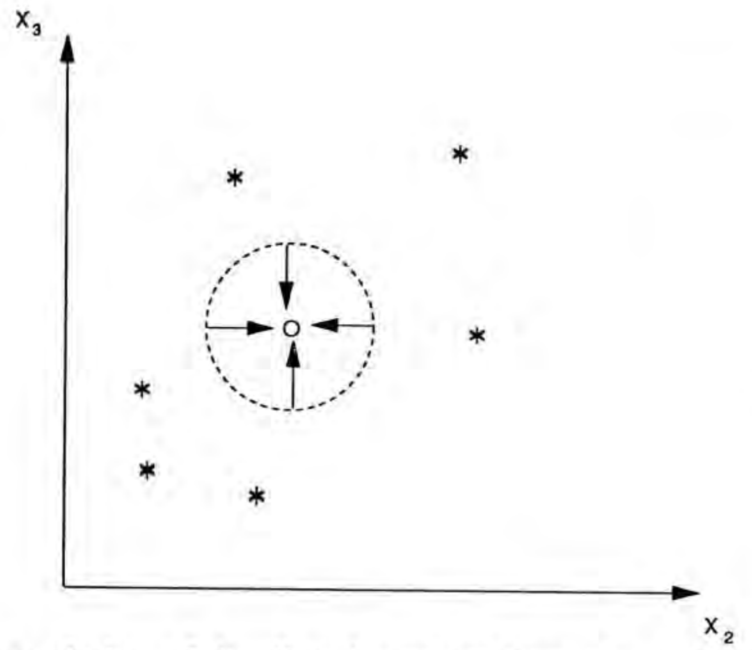

Fig 4.4 The solution of the disturbance problem by the RBF hidden unit

The arrows in fig 4.4 indicates those directions in which the radial basis function approaches its maximum value. We can now, by adjusting the height of the Gaussian functions through the output weight of the hidden unit, control the output value of the network at the new training pattern such that the error at the pattern is exactly compensated. If there is a stream of training patterns to be appended to the training set, each training pattern could be enclosed in such a radial basis function instead of four hyperplanes, which constitutes a resulting four-fold decrease in the network size.

Actually, we would not like to have each Gaussian function enclosing only one training pattern. It would be useful if some kind of iterative optimization process is incorporated into the above node addition scheme such that each hidden unit can cater for more than one training pattern. Unfortunately, there is no uniform standard for training RBF networks: a popular method, as exemplified by Tsoi [55], and Chen and Cowan [7], involves the pre-selection of all the centres of the RBF in the neural network, and then solves for all the output weights by linear least-square methods. This approach, however, requires the knowledge of a priori information on the training set. For example, in the context of classification, it would be ideal if we first apply some forms of clustering to the original training set, and select the centroid of each class as the centre for each RBF hidden node. The resulting network would give a reasonably faithful representation of the training set. Otherwise, with an arbitrary selection of the centres, the network is not guaranteed to represent correctly the underlying distribution of the data. An alternative method involves the adaptation of these centres of the RBF hidden units such that the initial misassignments can be corrected. In general, we can apply a BP-type optimization procedure to the network such that the centres and variance of each RBF unit can be independently adapted ie. we construct Δc_j and $\Delta\sigma_j^2$ such that they are proportional to the partial derivative of the total error with respect to c_j and σ_j^2. Assuming that we are using the Gaussian function as our RBF function: since the term c_j is in the numerator of the exponent of the exponential function and σ_j^2 is in the denominator, the partial differentiation would result in entirely different adaptation equations for c_j and σ_j^2. In addition, the parameters to be adapted include all the components of $\mathbf{c}$ and the σ_j^2 for all dimensions, thus resulting in a two-fold increase in the number of parameters to be adapted when compared with a conventional BP network. The over-abundance of parameters for a single hidden node ,though resulting in a higher convergence rate for RBF network, is also the source of over-fitting when applying the RBF network to tasks such as time series modelling as described in [58].

Therefore, it would be ideal if we can derive a method in which whenever a new training pattern is appended to the training set, it could be enclosed completely by a single new hidden unit. Moreover, an iterative optimization procedure has to follow the node assignment such that one hidden unit can in general handle more than one training pattern. The optimization process should generate a single type of adaptation equation for all the parameters of the hidden node, and the number of parameters to be adapted should not be more than a small fraction of those in the corresponding sigmoid network. It would be most ideal if we can directly apply the classical BP process to fine-tune the network. The above description seems an over-idealization, but the above combination actually exists in the form of the present growth algorithm to be described in this chapter.

4.3 The Additional Input Node and the Modified Nonlinearity

The ideal network promised in the previous subsection can be realized by simply appending an additional input node to each hidden node. The situation is depicted in fig 4.5.

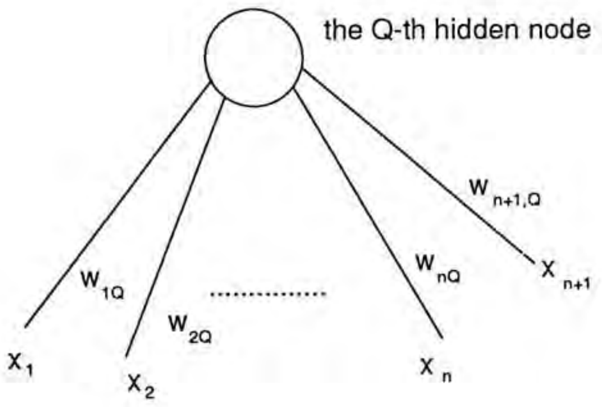

Fig 4.5 The architecture of the modified hidden node in the growth algorithm

We should now investigate the property of this modified hidden node. Defining the pre-nonlinearity activation a_Q for the Q-th hidden node as

$$a_Q=\sum_{i=1}^{n+1} w_{iQ}x_i \tag{4.3}$$

and defining

$$x_{n+1}=\sum_{i=1}^{n} x_i^2$$

The above summation becomes

$$\begin{aligned} a_Q&=\sum_{i=1}^{n} w_{iQ}x_i+\sum_{i=1}^{n} w_{n+1,Q}x_i^2 \\ &=-\sum_{i=1}^{n} b(x_i-c_i)^2+d \end{aligned} \qquad (4.4)$$

From the above pre-nonlinearity activation, we find that the output of the hidden node, h_Q, is equal to

$$\begin{aligned} h_Q&=f(a_Q) \\ &=\frac{1}{1+\exp(\sum_{i=1}^{n} b(x_i-c_i)^2-d)} \end{aligned} \qquad (4.5)$$

It can be seen that the output of the hidden unit depend only on the distance between the input **x** and a centre **c** with its components defined by the hidden weights of the modified hidden node. Thus the above function can be qualified as a radial basis function. The functional value at the centre of this radial basis function acts as either the maximum or minimum of the overall function depending on the sign of b.

The purpose of the derivation of this modified hidden node lies in its underlying simple weight adaptation strategy: if we ignore the modified functional form of the modified node and treat it simply as a black box, the modified node is simply a conventional sigmoid hidden unit with n+1 input nodes. The conventional BP algorithm can be directly applied to this unit without further modifications. Furthermore, the number of weights to be adapted is n+1, a negligible increase in the number of modifiable parameters when compared with a conventional radial basis function where 2n parameters are to be adapted.

In additions, one may notice that this new radial basis function contains two additional parameters, b and d, when compared with the conventional RBF function. The parameter b can be equated with the variance σ_i^2 of the traditional RBF, but in the present case the parameter b can take on positive or negative values depending on the hidden weights while the σ_i^2 only take on positive values. The resulting function thus can vary from the form in which the centre takes on the maximum value to another form in which the centre takes on the minimum value. Thus an additional degree of freedom is possessed by the current node. Furthermore, in conventional RBF, the value of d is always zero. When interpreted in the context of the current modified hidden node, this implies that all the hidden weights w_{iQ} must obey a certain relationship among themselves. On the other hand, the current adaptation scheme attempts to adapt all the weights w_{iQ} independently, resulting in a value of d which is normally non-zero. Thus, a further degree of freedom is added to the system.

Finally, there is no reason in the course of weight adaptation which prevents the weight $w_{n+1,Q}$ from attaining a near-zero value, which causes the modified nonlinearity of the node to revert to a normal sigmoid function. Thus the modified nonlinearity can in fact change continuously from a normal sigmoid function to a RBF function according to the requirement posed by the training set, and can thus elicit the advantages of these two kinds of nonlinearities.

4.4 The Initialization of the New Hidden Node

Suppose that in the course of dynamic node creation, the current restricted network can no longer cope with the difficulties of the current training set, and there exists some training patterns at which the representation error is non-negligible. It would be reasonable if we pick out one training pattern among these patterns which invokes the greatest error and creates a RBF hidden node to enclose this training pattern such that the reduction of error would be greatest. This node can be created by appropriately choosing the hidden weights of the modified node. First assuming that b is 1 to give a reasonable variance for the RBF and set d equal to zero to conform with the conventional RBF, we have

$$a_Q=-\sum_{i=1}^{n}(x_i-c_i)^2 \tag{4.6}$$

It is noticed that the sign of b is positive such that the resulting nonlinearity attains its maximum at the centre. Designating the pattern which possess the greatest error as the "difficult" pattern and symbolizing it as $\mathbf{x}_d$, we assign this pattern as the centre of the new hidden node, and the following expression results:

$$\begin{aligned} a_Q&=-\sum_{i=1}^{n}(x_i-x_{di})^2\\ &=-\sum_{i=1}^{n}(x_i^2-2x_{di}x_i+x_{di}^2)\\ &=-\sum_{i=1}^{n}x_{di}^2+2\sum_{i=1}^{n}x_{di}x_i-\sum_{i=1}^{n}x_i^2 \end{aligned} \tag{4.7}$$

Designating x_1 as the bias unit and comparing this expression with the activation of the modified node, we obtain

$$\begin{aligned} w_{1,Q}&=-\sum_{i=1}^{n}x_{di}^2\\ w_{i,Q}&=2\sum_{i=1}^{n}x_{di} \qquad 2\le i\le n\\ w_{n+1,Q}&=-1 \end{aligned} \tag{4.8}$$

The above initialization would result in a RBF which peaks at the "difficult" pattern and tails off gradually on all sides. The rate of tailing off is controlled by the magnitude of the various hidden weights. Suppose that the various desired output for the "difficult" pattern at the various output nodes are denoted by $d_1, \ldots\ldots, d_m$. Since the output of the new hidden node at the "difficult" pattern is 0.5 by the above equation, the output weight u_{Qk} of the new hidden node can be initialized as

$$u_{Qk} = \frac{f^{-1}(d_k) - \sum_{j=1}^{Q-1} u_{jk} h_j}{0.5} \tag{4.9}$$

where f^{-1} represents the inverse of the sigmoid function and h_j represents the output of the various hidden nodes. With all its parameters determined, we can apply the BP process to the appended network such that further optimization of the parameters are possible. At this stage we should define what is meant by the success of the BP-fine-tuning process: we would usually stop the BP process if the successive percentage decrease of error $\Delta E/E$, where $E = E(t+1)-E(t)$ and t denotes the number of epochs, is less than a pre-determined threshold G, which we call the error gradient threshold. We would then examine the mean square error of the resulting network: in a later subsection it would be proved that under the present node creation scheme, the final network will at most contain p - 1 hidden nodes, where p is the number of training patterns in the training set. With reference to this final condition, we could define two practical criteria to determine whether the BP fine-tuning process is successful:

(1) For a classification problem in which the training data is relatively noise-free (such as the Parity Problem), we would consider that the BP fine-tuning process has succeeded if one more training pattern (not necessarily the "difficult" pattern) previously not classified correctly is now classified correctly on the addition of the new node. (The definition of correct classification is such that the output of the network should be on the correct half of the interval [0, 1] if the desired output is 0 or 1)

(2) For a training set in which the existence of noisy data are suspected or for a function approximation task, we could not adopt the number of instances of correct classification as

our convergence criterion. Instead we should set a threshold error E_T such that if the error of the overall network is less than E_T, the network is said to have adequately represented the training set. Assuming also that we are at a stage of node creation in which the Q-th node is recently appended to the network: before the addition of the new node, the error of the network is E_i, while after the BP fine-tuning the error of the network is E_f. To fully represent the training set with a network of at most p-1 nodes as would be possible under the current growth algorithm, the new node should at least cause the error to decrease by an amount of $(E_i - E_T)/(p - Q)$. In other words, the decrease in error $E_i - E_f$ through the node addition and the BP process must satisfy $E_i - E_f \geq (E_i - E_T)/(p - Q)$. If this criterion is satisfied, the BP fine-tuning process is considered successful.

According to the nature of the training set, we would call the achievement of the above two criteria the satisfaction of the partial convergence criterion.

In view of the possibilities of unsuccessful instances for the BP process, we should prepare a precautionary measure for the new hidden node: prior to the BP fine-tuning process, the state of the network is recorded, and if the BP process is not successful, the previous state of the network is restored, and the following scale-up is applied to the hidden weights.

$$w_{iQ(new)}=sw_{iQ(old)} \tag{4.10}$$

The s in the above expression is the scale-up factor and should be greater than 1. This scale-up process is to be repetitively applied if the BP process is persistently unsuccessful. Since the parameter s, when substituted into the original equation, becomes synonymous with the variance of the RBF: if we let s approach infinity, the RBF would be so narrow that the disturbance of the hidden node to all the other training patterns (except the "difficult" pattern) would be zero, and in this way the error at the "difficult" pattern would be exactly compensated by the new hidden node without affecting the other training patterns.

4.5 Initialization of the First Node

It is seen in the previous subsection that the various parameters of the new hidden node is wholly dependent on the "difficult" pattern. Suppose that we are going to start a new training session and we possess a single-node network. Since the network has never been trained before and we would not favour a random initialization under the present scheme, the concept of "difficult" pattern is not defined and alternative initialization scheme has to be derived for the first node of the network. We would suggest here an initialisation scheme which when coupled with the preceding initialization scheme for subsequent hidden nodes, would result in a final network with at most p - 1 hidden nodes.

We would first select among all the training patterns the pair which possesses the greatest inter-pattern distance, and denote them by **x**(t) and **x**(t'). We would now introduce our initialization scheme for the first hidden node: the hidden weights of the node are given by:

$$\begin{aligned}
w_{1Q} &= -\frac{1}{2}\sum_{i=1}^{n}(x_i(t)-x_i(t'))(x_i(t)+x_i(t')) \\
w_{iQ} &= x_i(t)-x_i(t') \qquad 2\le i\le n \\
w_{n+1,Q} &= 0
\end{aligned} \tag{4.11}$$

The above initialization essentially places the hyperplane of the first node in the mid-point between the training patterns **x**(t) and **x**(t') with its normal vector aligned with the difference vector **x**(t) - **x**(t') between the two vectors. The relationship between the hyperplane of the first node and the domain of the training set is depicted in fig 4.6.

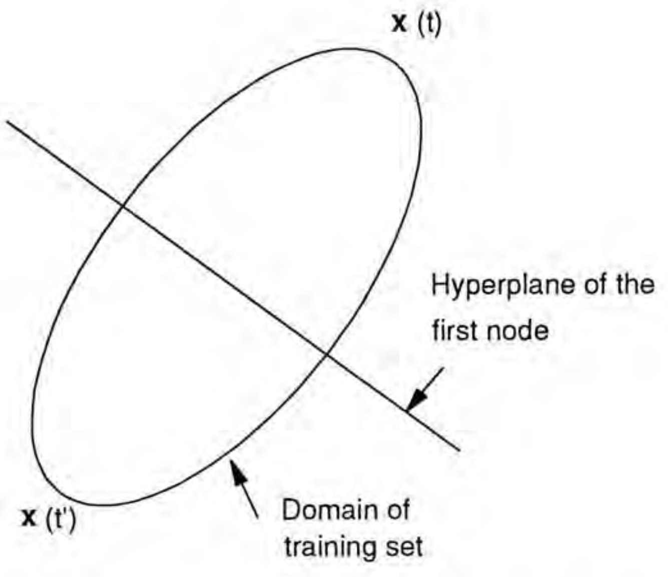

Fig 4.6 Relation between the hyperplane of the first node and the training set domain

It is seen in fig 4.6 that the hyperplane of the first node is placed almost across the centre of the training set domain and partitions the domain into half. Besides acting as a preparation step for the subsequent exact representation of **x**(t) and **x**(t'), this arrangement actually facilitates the BP fine-tuning process: as can be seen in the above diagram: no training patterns are at an excessively large distance from the hyperplane as the hyperplane is placed almost at the centre of the training set domain. In the context of the sigmoid nonlinearity, since the hyperplane is the site of the greatest gradient for the function, and no training patterns are excessively far from this site, this implies that at the various training patterns the gradient of the sigmoid function would not be excessively small. Since the adaptation term for the hidden weights in the BP equations is directly proportional to the gradient of the sigmoid function at the various training patterns, this implies that the adaptation term at the various training patterns for the first node would not be excessively small, and the BP process can make a real contribution in reducing the overall error of the network.

After the initialization of the hidden weights, we can further proceed to initialize the output weights u_{2k} of the first hidden node (after the bias node) such that the resulting network can exactly represent the two patterns. Denoting the desired output vector for the two pattern as **d**(t) and **d**(t'), and the corresponding hidden node output for the first node as $h_2(t)$ and $h_2(t')$ respectively (the subscript 1 is reserved for the bias node). We can solve for the output weight u_{2k} and the bias weight u_{1k} by the following pair of linear equations.

$$\begin{aligned} u_{1k}+h_2(t)u_{2k}&=d_k(t) \\ u_{1k}+h_2(t')u_{2k}&=d_k(t') \end{aligned} \tag{4.12}$$

Since there are two equations with two unknowns, we can exactly solve for the bias weight u_{1k} and the output weight u_{2k} of the first node. In this way the parameters of the first node can be fully initialized.

The criterion of success for the BP fine-tuning process is simple: we must remember that through the above initialization scheme, the single-node network can already represent the two patterns **x**(t) and **x**(t') exactly. We can simply store the current state of the network and apply the BP process until $\Delta E/E \leq G$ and check if the final error E_f is greater than the initial error E_i (a possibility since the pattern-update mode of BP is not an exact gradient process). If this is not the case, we can accept this solution as final, otherwise we can simply restore the previous state of the network.

Finally, one may notice that the searching for **x**(t) and **x**(t') is a very time-consuming process in that the distance between every pair of training patterns in the training set have to be obtained. In view of this, we have adopted a simplified and sub-optimal method in searching for **x**(t) and **x**(t'): Each component of each training pattern is shifted by a suitable constant amount such that all the components of every training pattern is positive (e.g. in a 2-D training set, all the patterns are shifted to the first quadrant). After that we calculate the Euclidean distances of all the shifted pattern and selecting the one with the smallest Euclidean distance as **x**(t') while selecting the one with the largest Euclidean distance as **x**(t).

4.6 Practical Considerations for the Growth Algorithm

It will be shown in subsection 4.7 that the present algorithm will at most generate p - 1 hidden nodes for every training set, thus satisfying the convergence requirement for a dynamic node creation algorithm. However, for a given task, we would like the present algorithm to generate hidden nodes in an effective manner such that the training set can be represented with as few hidden nodes as possible. The above node initialization scheme concerns primarily with the eventual convergence aspect of the algorithm while disregarding the efficiency issue of the node addition procedure. In addition, the BP fine-tuning process would be applied to the restricted network immediately after the node initialization procedure but there is no guarantee that the BP process would be well matched with the parameters evaluated from the deterministic assignment procedure, i.e. we cannot guarantee that the BP process would make an effective descent starting from the current new configuration of the network. Through experiments, we have discovered two heuristics which would ensure this effective descent after the deterministic assignment procedure. These will be described below:

(1) Magnitude Adjustment for $w_{n+1,Q}$: The additional weight $w_{n+1,Q}$ is an exclusive feature of the current growth algorithm such that a dynamic node creation algorithm can be built on an unordered training set. This weight is fed by the summation $x_{n+1}=\Sigma_i x_i^2$. Due to this operation the magnitude of x_{n+1} is usually much greater than the other input components $x_1,\dots\dots, x_n$. This fact is not in conflict with the convergency requirement but unfortunately in conflict with the BP fine-tuning process: since the hidden weight adaptation term in BP is directly proportional to the magnitude of the input components:

$$\Delta w_{ij}=\eta\delta_j x_i \tag{4.13}$$

The large magnitude of x_{n+1} will cause a correspondingly large adaptation to $w_{n+1,Q}$ when compared to other hidden weights. Under this condition the underlying RBF may not properly preserve its own shape and perform its task of isolating the "difficult" pattern during the BP fine-tuning phase. In fact, it is observed in experiments that the error decrease under the BP process when the above scheme is directly applied is not remarkable due to the reason that the large changes in $w_{n+1,\,Q}$ results in the underlying RBF not being able to properly cover the "difficult' pattern. The solution is simple for this problem: we can simply scale down x_{n+1} by a proper amount and scale up the weight $w_{n+1,\,Q}$ by the corresponding amount: Here we scale the input x_{n+1} according to the following scheme:

$$x'_{n+1}=x_{n+1}\times\frac{c_1}{c_2} \tag{4.14}$$

where

$$c_1=E[\,|x_i(t)|\,] \qquad \forall t,i$$
$$c_2=E[\,|x_{n+1}(t)|\,] \qquad \forall t$$

and the corresponding hidden weight $w_{n+1,\,Q}$ is scaled up as follows

$$w'_{n+1,Q}=w_{n+1,Q}\times\frac{c_2}{c_1} \tag{4.15}$$

In this way the magnitude of x_{n+1} is of the same order as the other input components x_i. Moreover, by the corresponding scale up in the hidden weight $w_{n+1,\,Q}$, the shape of the RBF established by the node initialization scheme is maintained. Experimentally it is observed that through the above scaling scheme, the performance of the BP fine-tuning process is greatly improved, and a true descent of the error surface is achieved.

(2) Monitoring of $\delta(\Delta E/E)$: As has been mentioned previously, the end of the BP fine-tuning process is determined by the quantity $\Delta E/E$ where E =E(t+1) - E(t) signifies the change in error over one epoch: the BP process is stopped whenever $\Delta E/E$ is smaller than the error gradient threshold G, indicating that the current state of the network has reached a certain region in the error surface at which the gradient approaches zero, signifying the possibiity of encountering the global minimum. This monitoring scheme performs satisfactorily during the intermediate stages of BP fine-tuning process when $\Delta E/E$ gradually decreases as expected,

but not at the stage immediately after a new node is appended to the network: the presence of the new node provides an alternative dimension through which the current state of the network can descend, but through experimentation it is discovered that the network may spend quite some time in searching for this alternative path, and during the searching stage it is observed that the value of $\Delta E/E$ remains low and may sometimes fall below the threshold G, indicating that the current state is on a plateau and is about to descend along the alternative path. In view of this, if the node addition scheme only monitors the value of $\Delta E/E$, it may misinterpret the situation immediately after the node addition as the signal for the completion of a descent and the requirement of a new node. Under this condition, many redundant nodes may be added to the network before a true descent on the error surface is achieved. Fortunately, there does exist one difference which distinguishes the cases between the start of descent on a plateau and the completion of a descent: if we calculate the quantity $\delta(\Delta E/E) = \Delta E/E(n+1) - \Delta E/E(n)$ for both cases, we will find that for the first case, this quantity will be positive due to the gradual increase in the gradient of the error surface as the current state gradually approaches the alternative descent path. For the second case, the quantity will be negative due to the gradual decrease in the gradient of the error surface as the current state approaches a minimum point. Therefore, if at any stage of the BP fine-tuning process we discover that $\Delta E/E \leq G$, we can, by monitoring the sign of
$\delta(\Delta E/E)$ in addition, distinguish the above two situations: if $\delta(\Delta E/E)$ is negative, we can terminate the fine-tuning process. If $\delta(\Delta E/E)$ is positive, we should continue with the fine-tuning process until $\delta(\Delta E/E)$ is negative, and repeat the above process of monitoring the two quantities. In this way the addition of redundant nodes to the network due to the above reason is avoided. This enhanced error gradient monitoring scheme will also be applied to the deterministic dynamic node creation scheme described in Chapter 6.

4.7 The Convergence Proof for the Growth Algorithm

At this subsection we will prove that the current growth algorithm will eventually result in a network with a finite hidden layer: the proof will be obvious in view of the various developments mentioned in the previous subsections concerning the initialization scheme of the new node.

For the first node, we have ensured that the node can represent exactly two patterns through the initialization scheme of the first node.

For the other hidden nodes, we have ensured through the hidden node initialization scheme that the resulting node can at least represent one training pattern exactly. There are p - 2 such patterns (minus the two patterns represented by the first node) to be catered for by these additional hidden nodes. Therefore, for the worst case in which the first node can represent only two patterns and each hidden node can only represent one pattern, the network consist of $1 + (p-2) = p - 1$ hidden nodes. Therefore, thee hidden layer size generated by the current algorithm will still be finite under the worst condition. It can be seen that the eventual network size of the current algorithm is the same as that for the progressive training algorithm [9].

4.8 The Flow of the Growth Algorithm

We may now recapitulate on the flow of the growth algorithm : the flow of the algorithm is depicted in the flowchart in fig 4.7

4.9 Experimental Results and Performance Analysis

1. The Parity Problem

The growth algorithm is first applied to the parity problem , which is long considered a very difficult problem because by changing only one bit of information, the output of the network would be entirely different. In the simulation studies, we adopt the adaptation gain of 0.5 and a momentum of 0.7. To minimize the training time, we adopt the following criterion of convergence, if the desired output is 1 for a training pattern, we would consider that the output of the network is correct if it is greater than 0.5. On the other hand, if the desired output is 0, the correct classification criterion would be that the network output is smaller than 0.5. The error gradient threshold G is set at 0.05% since this threshold results

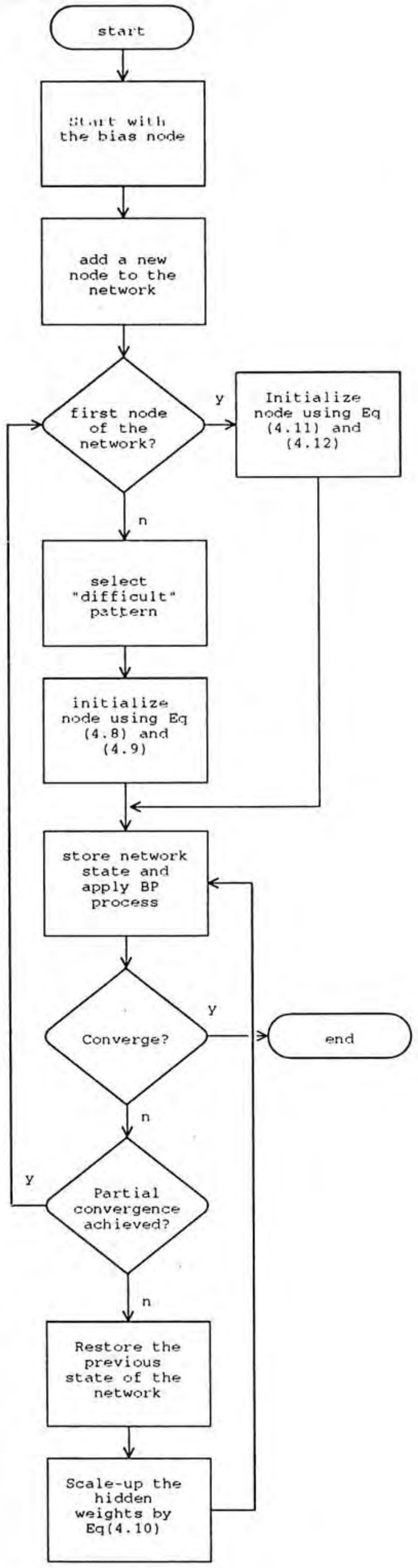
start
Start with the bias node
add a new node to the network
first node of the network?
y
Initialize node using Eq (4.11) and (4.12)
n
select "difficult" pattern
initialize node using Eq (4.8) and (4.9)
store network state and apply BP process
Converge?
y
end
n
Partial convergence achieved?
y
n
Restore the previous state of the network
Scale-up the hidden weights by Eq(4.10)

Fig 4.7 Flow of the growth algorithm

in a reasonable network size for parity problems of most orders. The scale-up factor for the growth algorithm is set at 5, 5.5, 6, 6.5, 7, and the simulation results presented below is averaged over these 5 values for the scale-up factor.

In addition, the progressive training algorithm is also applied to the same problem and the results are tabulated alongside the results for the growth algorithm. The parameters used for the progressive training algorithm is the same as that used for the growth algorithm and the results are averaged over the same 5 scale-up factors. Finally the BP algorithm is also applied to the same problem with its hidden layer size equals that for the growth algorithm for comparison purpose.

parity order	Growth Algorithm		PT Algorithm		BP Algorithm	
	No. of Epochs	No. of Nodes	No. of Epochs	No. of Nodes	No. of Epochs	No. of Nodes
2	39	2	87	2	107(0%)	2
3	64	3	165	3	62(0%)	3
4	124	4	373	5	698(80%)	4
5	674	4	489	11	565(20%)	5
6	579	9	382	14	92(0%)	9
7	782	10	1050	15	121(20%)	10

Table 4.1 The growth algorithm applied to the parity problem

The figures in the parentheses in Table 4.1 indicates the failure rate among 10 trials of BP. From the above results, we can summarize our observation.

(1) Like the Progressive Training Algorithm, the growth algorithm does not encounter any local minimum. While there are occasional failure rate among 10 trials of BP for each parity order, both the growth algorithm and the progressive training algorithm encounter zero failure

rate.

(2) For the convergence speed, the growth algorithm excels at both low-order and high-order parity problems, while the progressive training excels at parity problems of intermediate orders. On the other hand, for low-order parity problems, the convergence speed of BP is in general lower than that of the other two algorithms, while for the higher-order parity problems, the convergence speed of BP is much greater. This can be explained by the fact that we have used the same number of hidden nodes for the BP network as for the network built up using the growth algorithm such that a comparison between these two types of networks can be made. As in general, the parity problem of order n can be solved by a network containing n hidden nodes, we are in fact using more nodes than necessary in the BP case and thus results in a quick convergence. This also highlights one of the problems of dynamic node creation, that the convergence speed would be slow when compared with BP networks with a large initial hidden layer size. However, this is more than compensated by the fact that in the BP case, we have no means whatsoever for determining the network size and we cannot guarantee the convergence of the network as indicated by the non-zero failure rate.

(3) The network size built up by the growth algorithm is in general smaller than that for the progressive training algorithm and is much closer to the optimum size of n hidden nodes for the n-th order parity problem. In other words, the growth algorithm serves as a better hidden layer size estimator than the progressive training algorithm in this problem.

2. The Handwritten Character Recognition Problem:

The present algorithm was also applied to a hand-written character recognition problem. The input features are extracted through the segment projection approach, the details of which are reported in Lee et. al [31]. This feature extraction method results in 16 input nodes for the input layer. The character set consists of the alphabets A to Z and includes the numerals 0 to 9. The output layer consists of 36 output nodes with each of the nodes corresponding to the recognition of a particular character by presenting an output of

1 with the output of all other nodes at zero. The training set consists of 180 examples from the handwritings of 5 persons. The testing set consists of another 180 examples from the handwritings of another 5 persons. The training is terminated when the network sum-squared error E is less than 30, which results in a reasonably low classification error for most of the cases. The training parameters are identical to the parity case, while we have separately tabulated the training results for the various error gradient threshold G in order to investigate its relationship with the generalization capability of the network. The BP results are obtained from averaging 5 training trials. We first compare the training speed of the various algorithms.

Error Gradient Threshold	Growth Algorithm	PT Algorithm	BP Algorithm
0.05%	511	1685	83
0.06%	507	1733	83
0.07%	431	1202	83
0.08%	356	1480	50
0.09%	340	952	67
0.10%	349	1245	50

Table 4.2 The growth algorithm applied to the handwritten character recognition problem (Training speed comparison)

Since the concept of error gradient threshold is not applicable to the BP case, the results for BP in Table 4.2 is obtained simply by using a network with the same hidden layer size built up from the growth algorithm using the corresponding error gradient threshold, thus resulting in the identical simulation results for BP across several values of error gradient threshold, since identical network size are generated by the corresponding growth algorithm.

It is noticed that the convergence speed for BP is in general much higher than that the dynamic node creation algorithms due to its initial large network. Thus the defect of starting with a single-node network for the node addition algorithm is once more highlighted. In general, the convergence speed of the growth algorithm is greater than that for the progressive training approach due to the special training sequence adopted by the latter algorithm in which the training set has first to be segregated and then reassembled pattern by pattern during the course of training ,thus resulting in a lower speed of convergence. Moreover, the convergence speed decreases as the error gradient threshold increases for the two algorithms, which may be due to the reason that the high error gradient threshold encourages a quick build-up of hidden nodes which in turn removes quickly the initial single-node network restriction. We would next compare the resulting network size generated by the two dynamic node creation algorithms:

Error Gradient Threshold	Growth Algorithm	PT Algorithm	BP Algorithm
0.05%	18	28	18
0.06%	18	26	18
0.07%	18	24	18
0.08%	22	27	22
0.09%	21	25	21
0.10%	22	32	22

Table 4.3 The growth algorithm applied to the handwritten character recognition problem (Network size comparison)

It can again be seen that the growth algorithm produces a smaller hidden layer size than the progressive training algorithm. In general, the number of hidden nodes generated by the two algorithms increases when the error gradient threshold is increased which corresponds to a relaxation of the node addition criterion.

The recognition rate for the various algorithms on the training set is then compared:

Error Gradient Threshold	Growth Algorithm	PT Algorithm	BP Algorithm
0.05%	93.3%	97.2%	92.7%
0.06%	92.7%	98.9%	92.7%
0.07%	92.7%	96.7%	92.7%
0.08%	82.2%	97.2%	94.4%
0.09%	88.3%	97.8%	91.1%
0.10%	85%	97.2%	94.4%

Table 4.4 The growth algorithm applied to the handwritten character recognition problem (Training set recognition rate)

Since we are not pursuing 100% classification in the training exercise, complete classification is generally not achieved for most of the cases. However, the classification rate on the training set for all of the 3 algorithms are relatively high, in some cases approaching 100%. In general, the progressive training algorithm achieves the highest classification rate due to the near individual catering of each training pattern and its long training time. In general, the classification rate for both the growth algorithm and BP algorithm is similar since the complete training set is presented to the network in every trial which results in a generalization over the whole training set rather than the fitting of the individual patterns. This results in a slightly lower classification rate for these two algorithms but the underlying natural training sequence can help to prevent overfitting. Finally, it is noticed that the classification rate for all three algorithm does not vary much across various error gradient threshold.

Finally, the recognition rate of the networks produced by the various algorithms on the test set is compared:

Error Gradient Threshold	Growth Algorithm	PT Algorithm	BP Algorithm
0.05%	81.1%	72.8%	83.3%
0.06%	82.8%	71.7%	83.3%
0.07%	86.1%	74.4%	83.3%
0.08%	80.6%	76.7%	88.9%
0.09%	82.7%	78.3%	85.6%
0.10%	85.0%	71.1%	88.9%

Table 4.5 The growth algorithm applied to the handwritten character recognition problem (Test set recognition rate comparison)

For the generalization capability on the test set, the conventional BP algorithm exhibits a higher generalization rate than either of the two dynamic node creation algorithms, which again confirms the notion that in a dynamic node creation environment, the knowledge of the training set may already be consolidated in a small network at the early training stages. This initial small network may be inadequate for that particular training set when compared to a large network. In addition, the generalization rate of the growth algorithm is greater than that for the progressive training algorithm due to the shorter training time involved which reduces the chance of overfitting. In general, it is observed that the generalization rate is related to the convergence speed which in turn is related to the error gradient threshold ,rather than directly to the threshold itself. For example, the highest generalization rate for the two node addition algorithms occurs in the intermediate threshold range, which corresponds to the intermediate range of training speed where the possibility of overfitting and underfitting is slight.

(3) Time Series Modelling

For this simulation study, we adopt the Mackey-Glass time series as our training set. The time series is generated from the Mackey-Glass differential equations

$$\frac{dx(t)}{dt}=\frac{0.2x(t-17)}{1+x^{10}(t-17)}-0.1x(t) \tag{6.19}$$

The equation was integrated using a fourth-order Runge-Kutta method to provide values of x at discrete time steps. The task is to predict x(t+6) from four past data points x(t), x(t-17), x(t-34), x(t-51). Thus networks with four input nodes and one output node were employed. In this experiment, the first 400 points of the series were used for training and the following 543 points were reserved for testing purpose. Thus the generalization performance of the PT algorithm can be tested. The normalized root-mean-square error (NRMSE) was used as a generalization performance index:

$$NRMSE=\frac{E[(d_1(t)-y_1(t))^2]^{0.5}}{E[(y_1(t)-E[y_1(t)])^2]^{0.5}} \tag{4.17}$$

where $d_1(t)$ is the target value x(t+6) in the testing data set and $y_1(t)$ is the network's prediction.

The training parameters used are the same as the previous two simulation studies. The error threshold used is 0.1 which provides a reasonable prediction error for all the training exercises. The training speed of the various algorithms is first compared.

Error Gradient Threshold	Growth Algorithm	PT Algorithm	BP Algorithm
0.05%	1129	1015	3523
0.06%	891	1177	2824
0.07%	814	1029	1505
0.08%	787	817	2824
0.09%	785	970	2465
0.10%	754	790	2465

Table 4.6 The growth algorithm applied to the time series modelling problem (Training speed comparison)

For this training exercise, it is discovered that the convergence speed for BP is lower than that for the two dynamic node creation algorithms, indicating that this training set, unlike the handwritten character recognition problem, is not at all a simple training data set for BP ,even when it is equipped with the advantage of starting with multiple nodes. The convergence speed for the two node addition algorithms are similar and exhibit the characteristic trend of increasing convergence speed for increasing error gradient threshold. The size of the network generated by the various algorithms is then compared:

Error Gradient Threshold	Growth Algorithm	PT Algorithm	BP Algorithm
0.05%	5	9	5
0.06%	8	11	8
0.07%	7	10	7
0.08%	8	10	8
0.09%	11	12	11
0.10%	11	12	11

Table 4.7 The growth algorithm applied to the time series modelling problem (Network size comparison)

These simulation results are consistent with the previous two problems in that the resulting hidden layer size for the growth algorithm is smaller than that for the progressive training algorithm. In addition , the results display the characteristic trend of an increasing hidden layer size with increasing error gradient threshold. This is due to the shortened BP fine-tuning time applied to the network when a higher error gradient threshold is used. As a result, a new hidden node is not adequately trained to realize its full error reduction potential, and additional nodes are required to compensate this reduction in its capacity.

Finally, the generalization capability of the networks generated by the various algorithms is compared.

Error Gradient Threshold	Growth Algorithm	PT Algorithm	BP Algorithm
0.05%	0.3213	0.0997	0.1011
0.06%	0.2439	0.1006	0.1013
0.07%	0.2407	0.1028	0.1001
0.08%	0.2391	0.1018	0.1013
0.09%	0.2411	0.0982	0.1092
0.10%	0.2536	0.0999	0.1092

Table 4.8 The growth algorithm as applied to the time series problem (NRMSE comparison)

It is seen that, for this training set, the generalization capability of the growth algorithm is not as high as that for the other two algorithms. The reason may be due to the initialization scheme of the growth algorithm which depends on a single training pattern: though the scheme for the progressive training algorithm also depends on a single training pattern, the periodic disturbance to the network by the continuous addition of new patterns would prevent the early consolidation of the training set knowledge in a small network. Apparently, the possibility of memorizing noisy training patterns is higher for the growth algorithm than for the PT algorithm.

In general, the growth algorithm provides a greater convergence rate for most of the training set in the simulation studies, due to the immediate "covering" of the "difficult" training pattern by the modified nonlinearity of the new hidden node. In addition, the resulting network built up by the growth algorithm is smaller in size when compared to the progressive training algorithm, since for the PT algorithm each new pattern added to the training set would have the potential of creating a new node for the network, which greatly contributes to the possibility of node addition. On the other hand, the growth algorithm does not require this segregation of the training set and thus the possibility of node creation is lowered. However, the generalization rate for the growth algorithm is not satisfactory for noisy training sets such as time series, due to its single-pattern node initialization scheme which encourages the memorization of noisy training patterns. It will be seen in Chapter 6 that this restriction is partly alleviated by the deterministic training algorithm.

4.10 Concluding Remarks

We have achieved the first stage of our promised course of algorithm development in the form of the current growth algorithm which builds a dynamic node creation process on an unordered training set. This is important for a training scheme which is capable of accommodating new information since by the very definition of a scheme involving a training set ordered according to the Euclidean distance, we must have all the training patterns available prior to the commencement of training such that we can apply the sorting procedure to the training set. The conception of a node addition scheme on an unordered training set thus paves the way for a future training algorithm which can achieve incremental learning. The current growth algorithm achieves this task by including an additional input node for each hidden node which emits the Euclidean distance of each training pattern such that the underlying nonlinearlity effectively becomes a radial basis function, which can in turn completely isolate the so called "difficult" patterns from the other patterns and control their errors: a task which is impossible for the progressive training algorithm to achieve except with four sigmoid hidden units. Moreover, the present algorithm is guaranteed to terminate with a finite network which is an essential requirement for any dynamic node creation algorithms with no pruning applied. The current algorithm is applied to the famous parity problem ,a handwritten character recognition task and a time series modelling task. It is observed that in general the performance of the growth algorithm is comparable to the progressive training algorithm both in terms of training speed and the final network generated. Unfortunately, it is seen that the generalization performance of the algorithm on some data sets is not satisfactory when compared with those networks trained using the conventional BP approach. It will be seen in later chapters that in fact the unsatisfactory generalization performance is a general feature of dynamic node creation algorithms (especially for those node creation schemes in which the complete training set is used as opposed to algorithms in which a partial training set is used such on P.T.) due to the early consolidation of knowledge in a restricted network which is too small to adequately represent all the essential features of the current training set. However, this problem will find relief in the methods proposed in Chapter 7 and 8.

Finally, we may notice that the current growth algorithm is still based on a single training pattern. It has been mentioned that for some training sets, the training patterns may be contaminated with noise, and when incidentally one of these patterns is selected as the "difficult" pattern and used for the initialization of a new hidden node, this noisy pattern will most probably be memorized by the new hidden node and cause an overall misrepresentation of the training set, since the noise is not a part of the features possessed by the training set. The solution to this problem is to initialize the hidden node by large number of training patterns so that the noisy effects can be averaged out. The conception of this initialization scheme is not at all easy to carry out in the domain of the training set where the training patterns are independent entities such that the correlation between them are difficult to visualize. However, it will be seen in Chapter 5, that just by a simple variation in the vector concatenation scheme of the training patterns, we will arrive at the T-vector concept which is conducive to the achievement of the purpose stated above.

5 KNOWLEDGE REPRESENTATION IN NEURAL NETWORKS

5.1 An Alternative Perspective to Knowledge Representation in Neural Networks: The Temporal Vector (T-Vector) Model

To facilitate the representation of the various parameters in neural networks, the associated parameters are often concatenated into a vector. Notable examples are weight vectors which are the connection of the input to a node (whether output or hidden node), the input vector which contains all the input to the network, hidden vector which are the output of all the hidden nodes, and those of all output nodes as output vector. These conventions originate by considering the neural network as a mapping device from $\mathbf{R}^n$ to $\mathbf{R}^m$, in which the input nodes and the output nodes naturally cluster into vectors into their respective **R** spaces. In other words, these concatenations facilitate the description of the mapping pairs spatially. The concatenation for the hidden node output merely follows the convention of the input layer and the output layer, and describes vectors in a fictitious domain known as the hidden space.

Until recently, the above model has been employed to study the behaviour of neural networks. The hidden weight vectors are viewed as hyperplanes in separating the input patterns into clusters, and the hidden space is recognised as the site at which the elements of the input space are mapped into such that they are more separable by a hyperplane (in the form of the output weight vector). To gain an understanding of the workings of the neural networks, one cannot prevent oneself from investigating the distribution of patterns in the hidden space. However, the determination of the optimum hidden vectors in the hidden space under this model is by no means a simple task as the optimality criterion, that the hidden vectors be linearly separable, is an extremely loose one which results in a large number of possible configurations. In fact, the searching for the optimal hidden vector is not a trivial task and turns out to be at least as difficult in determining the network mapping itself, which calls for the usage of iterative learning procedure in determining the network parameters instead of deterministic assignment in the first place. As a result, we can gain little information from the above spatial model in either enhancing the hidden layer representation or inspiration in improving the current learning procedures.

In this chapter a slight alteration in the concatenation of the various node outputs for describing the network is discussed and it will be shown that such a model will lead to a great enhancement to our understanding of the knowledge representation process in neural networks.

5.2 Prior Research Works in the T-Vector Approach

Clearly realizing the difficulties of the spatial vector approach, researchers have turned to the temporal vector or T-vector approach to find an alternative perspective in order to probe into the inner workings of the neural network. This new approach is still in its germinating stage as it takes times to turn researchers from working in their favourite S-domain into a somewhat less familiar and less intuitive T-domain. But the trend of domain switching is evidently in the ascendancy in view of the increasing amounts of works turned out in recent years which made use of this T-vector approach, due to the great simplicity offered by the current approach in visualizing the hidden representation of the neural network. For example, Chen et. al [7] employed this technique for selecting the "centres" of the RBF for a RBF network. They concatenated the whole output history of a RBF node through a sweep of the training set into what is essentially a hidden T-vector. Each training pattern or S-vector in the training set, if selected as the "centre" of the RBF hidden unit, would generate a unique hidden T-vector through the action of the RBF, and thus a p-pattern training set would correspond to a pool of p candidate hidden T-vectors (which the authors referred to as the regressors) for the RBF network. The authors attempted to draw from these p candidates q hidden T-vectors with q<P which correspond to q RBF hidden units for the network. They achieved this by selecting those hidden T-vectors which, when added to the RBF network, produces the greatest reduction in the unexplained variance of the network output and thus provides the greatest error reduction. Fujita [14] also derived an algorithm

which produces a new hidden T-vector which approximates the error T-vector of the network. However, since the author resorted to the use of exhaustive search for the new hidden T-vector, the application of the network is restricted to binary training set which provides a finite set for the exhaustive search procedure to operate on. Barmann et. al [2] went a step further by adopting an iterative procedure to adapt the hidden weights of each hidden unit in turn such that the corresponding hidden T-vector of each unit align themselves with the error T-vector of the network at that moment to as close an extent as possible. This algorithm will be further described in Chapter 6 in order to extend the embryonic ideas concerning T-vectors in this algorithm into the full formalism required in describing the various spaces spanned by the T-vectors in a neural network, most notably the input space **X**, the hidden space **H** and the inverse desired output space $\mathbf{D}^{-1}$.

5.3 Formulation of the T-Vector Approach

It is well known that the training set presents itself as a steady stream of input patterns $\mathbf{x}(t)$, t=1 to p, with components $x_i(t)$, i=1 to n, and a stream of desired output patterns $\mathbf{d}(t)$, t=1 to p, with components $d_k(t)$, k=1 to m. Our strategy involves the concatenation of the components with a fixed spatial index i or k into a vector for all t. In this way, we obtain n vectors of dimension p, $\mathbf{x}_i$, i=1 to n for the input layer, and m vectors $\mathbf{y}_k$, k=1 to m for the output layer, with the corresponding desired output T-vector denoted as $\mathbf{d}_k$, k=1 to m. Similarly, we can treat the various node outputs of the hidden layer in a similar way. Thus, if the network possesses q hidden nodes in the hidden layer, we can concatenate these node outputs into q p-dimensional vectors $\mathbf{h}_j$, j=1 to q in which the t-th component of each vector $h_j(t)$ represent the hidden node output due to the excitation of the t-th input pattern. As the components of these new vectors are temporally related, we will hereafter call these vectors the temporal vectors (or their abbreviated form T-vector) as opposed to the spatial vectors (or S-vectors) introduced at the beginning of this chapter.

This slight alteration in the arrangement scheme of the vectors nevertheless allows us to draw much information from the field of linear algebra in enhancing the description of the knowledge representation scheme in neural networks. In particular, the notion of linear independency is particularly important to our further discussion of T-vector space and would now be introduced:

Definition: A set of vectors $\mathbf{v}_i$, i=1 to n is said to be linearly independent if and only if $\alpha_1\mathbf{v}_1+ \ldots\ldots+\alpha_n\mathbf{v}_n=0$ implies $\alpha_i=0$, i=1 to n.

Assuming that we have a set of linearly independent vectors $\mathbf{v}_i$, the set containing their linear combinations $\alpha_1\mathbf{v}_1+\ldots\ldots+\alpha_n\mathbf{v}_n$, for all $\alpha_i\varepsilon\mathbf{R}$ constitutes what is so called a vector space. The most prominent characteristic of a vector space is that it has a unique zero element **0** in accordance with our definition of linear independence.(There are other properties that a vector space possesses, but for vectors $\mathbf{v}_i$ in the Euclidean space $\mathbf{E}^n$ they are automatically satisfied). The set of vectors $\mathbf{v}_i$ is then said to span the vector space **V**, and the dimension of this vector space is n.

It is now quite plain to see why we have adopt the alternative concatenation scheme: the input T-vector $\mathbf{x}_i$, i=1 to n, spans an input vector space **X** of dimension n, assuming that they are linearly independent. Similarly, the output T-vectors $\mathbf{y}_k$, k=1 to m, spans an output vector space **Y**, the hidden T-vectors $\mathbf{h}_j$,j=1 to q, spans the hidden space **H**, and the desired output T-vectors $\mathbf{d}_k$, k=1 to m, spans the desired output vector space **D**. Of course, we cannot ensure the linear independency of the input and output T-vectors since they are drawn from an external source - our training environment. But this condition is of slight importance for T-vectors in the input and output T-vector space as the linear dependency among these vectors merely results in a finite increase in the number of input dimension n'>n, and a finite increase in the number of output dimension m'>m, as the format of our training set consists only of training patterns of finite dimension.

However, the situation is different for the hidden space **H** spanned by the hidden T-vectors $\mathbf{h}_j$,j=1 to q. Though we are supplied with only a finite number of hidden nodes we do not have any a priori information on how we should select the size of the hidden layer. In other words, we do not know the dimension q of the hidden space **H**. In fact, this problem is one of those defects which hinder the acceptance of BP in its original form as a practical training algorithm. Phrased in our new terminologies, inappropriate selection of the hidden space **H**, either in orientation or dimension, will result in the neural networks not being able to adequately represent the training set.

In order to alleviate the above problem, various dynamic node creation method [9,12,13,34], have been adopted as seen in previous chapters such that the dimension of the hidden space can be increased if necessary. The issue of linear independency is particularly relevant here as any new hidden T-vector is generated by the node creation algorithm and in principle the algorithms can generate an infinite number of hidden T-vectors. Therefore, it is important that the new hidden T-vector should be linearly independent from the old hidden T-vectors in the original network, since otherwise any information contained in the new hidden T-vector would be embedded in the old set of hidden T-vectors and the node addition process would in principal go on indefinitely. Fortunately, the linear dependency of the new hidden T-vector is under our control as this T-vector is not only a function of the training patterns (of which we have no control over), but also a function of the hidden weights of the new node (the design of which is the chief objective of the various node creation algorithms). In the next chapter, we would formulate a hidden weight design strategy such that the resulting new hidden T-vector is always linearly independent from the old hidden T-vectors.

Finally, we would like to have a method to check for the linear independency of a set of T-vectors. The Gram-Schmidt orthogonalization procedure from the field of linear algebra readily provides this function:

The Gram-Schmidt Orthogonalization Procedure: Let $\mathbf{v}_i$, i=1 to n be a set of vectors spanning the space **V**. The Gram-Schmidt procedure transforms the $\mathbf{v}_i$'s into an orthogonal set of vectors $\mathbf{u}_i$'s such that the new vectors span the same space **V**. The procedure is carried out as follows:

Let $\mathbf{u}_1=\mathbf{v}_1$

For $2 \leq i \leq n$

$$\textit{define } \boldsymbol{u}_i=\boldsymbol{v}_i-\boldsymbol{P}_{i-1}\boldsymbol{v}_i \tag{5.1}$$

where $\mathbf{P}_i$ is the projection operator defined by

$$\boldsymbol{P}_{i-1}\boldsymbol{v}_i=\sum_{j=1}^{i-1}\frac{<\boldsymbol{v}_i,\boldsymbol{u}_j>}{\|\boldsymbol{u}_j\|^2}\boldsymbol{u}_j \tag{5.2}$$

and <.> is the usual scalar product.

It is obvious therefore that if a vector $\mathbf{v}_k$ is linearly dependent on the preceding set of vectors $\mathbf{v}_1,\ldots, \mathbf{v}_{k-1}$, the corresponding orthogonal vector $\mathbf{u}_k$ would be zero vector $\mathbf{0}$, since in this case the projection of $\mathbf{v}_k$ on the space spanned by $\mathbf{v}_1,\ldots, \mathbf{v}_{k-1}$ (or equivalently $\mathbf{u}_1,\ldots, \mathbf{u}_{k-1}$), i.e., $\mathbf{P}_{k-1}\mathbf{v}_k$ would be equal to $\mathbf{v}_k$ itself. Therefore, $\mathbf{u}_k=\mathbf{v}_k-\mathbf{P}_{k-1}\mathbf{v}_k=\mathbf{v}_k-\mathbf{v}_k=\mathbf{0}$. In this way we have an efficient method for checking the linear independency of the hidden T-vectors at our disposal, which is to be applied at every stage of node addition.

In general, the assurance of the linear independency of the hidden T-vectors is not an adequate guarantee that the knowledge embedded in the training set would be efficiently represented by the hidden layer. The hidden space **H** must also satisfy several relationships with the input space **X** and the desired output space **D** such that the above purpose can be achieved. This will be the topic to be discussed in the next subsection.

5.4 Relation of the Hidden T-Vectors to the Output T-Vectors

At this stage, it is appropriate for us to define several new terms:

$\mathbf{D}^{-1}$, the space spanned by the vectors $\mathbf{d}_k^{-1}$,k=1 to m, where $\mathbf{d}_k^{-1}=f^{-1}(\mathbf{d}_k)$ and f is the sigmoid function.

$\mathbf{Y}^{-1}$, the space spanned by the vectors $\mathbf{y}_k^{-1}$, k=1 to m, where $\mathbf{y}_k^{-1}=f^{-1}(\mathbf{y}_k)$.

$\mathbf{H}^{-1}$, the space spanned by the vectors $\mathbf{h}_j^{-1}$, j=1 to q, where $\mathbf{h}_j^{-1}=f^{-1}(\mathbf{h}_j)$.

Assuming now that we have a neural network which contains q hidden nodes in its hidden layer and assuming that the hidden T-vectors are linearly independent from each other. The resulting hidden space **H** spanned by these vectors are q-dimensional. At the same time, the network possesses m output nodes, for which our desired output T-vector $\mathbf{d}_k$, k=1 to m is defined for each output node. We can apply the inverse sigmoid function f^1 to each output T-vector to obtain a set of new T-vectors ,the inverse desired output T-vectors $\mathbf{d}_k^{-1}$, k=1 to m, where $\mathbf{d}_k^{-1}=f^1(\mathbf{d}_k)$. Each component of each T-vector $\mathbf{d}_k^{-1}$ represents the desired pre-nonlinearity summation of each output node.

In order to represent the training set exactly, each T-vector $\mathbf{d}_k^{-1}$ must be some linear combinations of the hidden T-vectors, i.e., $\mathbf{d}_k^{-1}=ß_1\mathbf{h}_1+.....+ß_q\mathbf{h}_q$ for all k and for some $ß_j$, j=1 to q. In other words, each $\mathbf{d}_k^{-1}$ must be a member of the hidden space **H** spanned by the hidden T-vectors $\mathbf{h}_j$. This leads to the following representation criterion:

Neural Network Representation Criterion: Any training set can be exactly represented by a neural network provided that $\mathbf{d}_k^{-1}\in\mathbf{H}$ for all k, i.e., when $\mathbf{D}^{-1}$ is a subspace of **H**.

At the same time, each $\mathbf{d}_k^{-1}$ is a member of the p-dimensional Euclidean space $\mathbf{E}^p$, since each $\mathbf{d}_k^{-1}$ contains p components. In other words, $\mathbf{D}^{-1}$ is a subspace of the Euclidean space $\mathbf{E}^p$. Therefore, the above representation criterion is satisfied when $\mathbf{H}=\mathbf{E}^p$. It is well known that $\mathbf{E}^p$ can be spanned by p linearly independent vectors of dimension p. As a result, the above representation criterion can be satisfied if we include p hidden nodes with each node emitting a linearly independent T-vector from the other hidden nodes. Since we normally include a bias node for the hidden layer, we can see that we need at most p-1 hidden nodes for the exact representation of any training set with p patterns. This situation is depicted in Fig. 5.1.

However, the dimension p of each T-vector is usually a large number in view of the size of practical training set which usually contains several hundred patterns or more. To fully represent these training sets, we would have to make use of networks with several

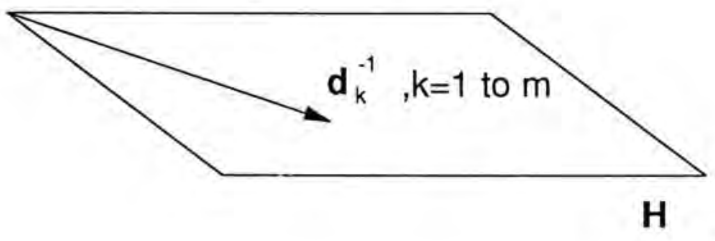

Fig 5.1 Condition for Exact Training Set Representation in Neural Networks

hundred hidden nodes or more. This is clearly not practical in view of the usual hidden size which contains only a small fraction of the above mentioned number of nodes. To compromise this inexact representation of the training set due to the inadequate number of nodes, we would endeavour to align the hidden space **H** such that the mean square error between all the T-vectors $\mathbf{d}_k^{-1}$ and their corresponding linear combination $\mathbf{d}_k^{-1\prime}=\beta_1\mathbf{h}_1+......+\beta_r\mathbf{h}_r$ where r<p is minimized. When interpreted geometrically, the above condition means that the distance of all the $\mathbf{d}_k^{-1}$'s to the hidden space **H** is minimized, i.e., that the $\mathbf{d}_k^{-1}$'s are closest to **H** as shown in Fig. 5.2.

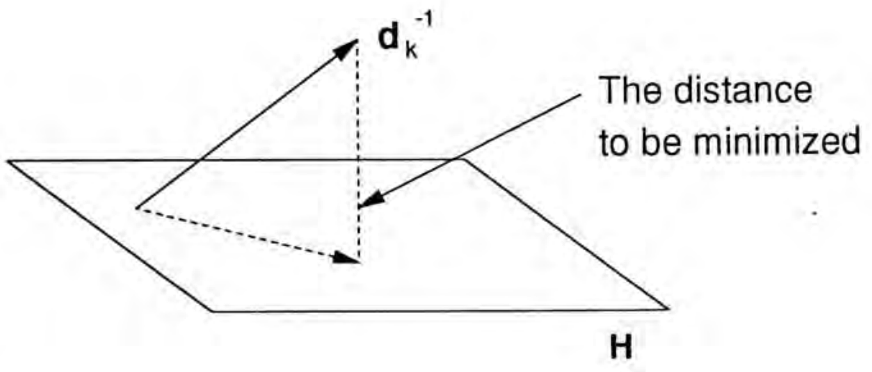

Fig 5.2 Suboptimal Representation of Training Set by a Small Network

In general, the above objective has to be achieved in two stages:

(1) The orientation of the hidden space H is adjusted using a network training algorithm (such as BP) such that the space **H** is reasonably close to all the $\mathbf{d}_k^{-1}$'s.

(2) At each stage of training, the projection of all the $\mathbf{d}_k^{-1}$'s onto the hidden space **H** (denoted by $\mathbf{P}_H\mathbf{d}_k^{-1}$) is evaluated and the corresponding distance between $\mathbf{P}_H\mathbf{d}_k^{-1}$ and $\mathbf{d}_k^{-1}$ (denoted as $\|\mathbf{d}_k^{-1}-\mathbf{P}_H\mathbf{d}_k^{-1}\|^2$) is calculated in order to evaluate the performance of the network. The projection operator $\mathbf{P}_H$ is defined as

$$P_H v=\sum_{j=1}^{q}\frac{<v,h_j^o>}{\|h_j^o\|^2}h_j^o \qquad h_j^o\in H \quad \forall j \tag{5.3}$$

where the T-vectors $\mathbf{h}_j^o$ are the orthogonalized version of the hidden T-vectors $\mathbf{h}_j$ obtained using the Gram-Schmidt orthogonalization procedure.

For conventional training algorithms such as BP, the second step is implicitly included in the first step, as the projection coefficients (depicted as the normalized scalar product term in Eq(5.3)) are determined by the algorithms in the form of output weights u_{jk} together with the hidden space orientation (in the form of hidden weights w_{ij}). However, for a dynamic node creation training algorithms, the above equation will be useful in determining the new output weight for the new hidden node. Several researchers [2,12] have exploited this possibility while still relying heavily on iterative procedures in determining the orientation of the hidden space **H**. It seems that only slight advantages have been gained in using the new concatenation approach since we are still unable to assign an a priori orientation to the hidden space even quasi-deterministically. However, the hidden space **H** is derived chiefly from the input space **X**, and we may as well probe into their relationships such that we can gain some insights into the method of assigning a favourable orientation to the hidden space through the input space, of which we have greater control over. This will be the topic of the next subsection.

5.5 Relation of the Hidden T-Vectors to the Input T-Vectors

The hidden space **H** is related to the input space **X** through the sigmoid function f. As seen in previous chapters, the sigmoid function is depicted as a monotonic increasing function which gradually flattens at both ends. However, the central portion of the function closely approximates a straight line. The deviation from linearity is slight even for the relative extreme functional values at $f(x)=0.2$ and $f(x)=0.8$. We can exploit this property in studying the relationship between the hidden space **H** and the input space **X**.

In principle, we can hardly visualize any relationship between the input space and the hidden space since they are related by a nonlinear transformation whose behaviour is in general not predictable. However, in view of the above described properties for the sigmoid function, and assuming we have some a priori information on the hidden T-vectors such that most of the components in these T-vectors do not exceed 0.8 or fall below 0.2 (this is not an unreasonable assumption as any practical training algorithms which drive the hidden vectors into deep saturation would severely limit their degrees of freedom and thus curtailing the ability of the neural network as a whole in representing any training set). Under this assumption the sigmoid nonlinearity can be well approximated by a straight line and theories of linear algebra can be employed to estimate the relationships between the two spaces. In other words, any hidden vectors $\mathbf{h}_j$ must be reasonably close to the input space $\mathbf{X}$ in order to be well approximated by it, since the sigmoid function is not a radical nonlinear function which can transform the input space X into any shape to suit the hidden space $\mathbf{H}$. The above situation is depicted in the following diagram:

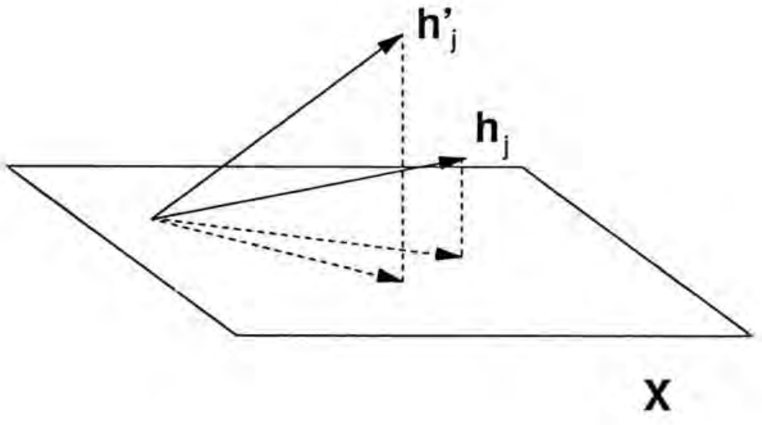

Fig 5.3 The Relationship between the Input Space $\mathbf{X}$ and the Hidden T-Vectors $\mathbf{h}_j$

From the above diagram, if we assume that we have two hidden spaces $\mathbf{H}$ containing the T-vectors $\mathbf{h}_j$,j=1 to q, and $\mathbf{H'}$ containing the T-vectors $\mathbf{h}_j'$,j=1 to q which represents the space $\mathbf{D}^{-1}$ equally well, we can see that the $\mathbf{h}_j$ vectors are more adequately represented by $\mathbf{X}$ than the $\mathbf{h}_j'$ vectors due to their smaller distances from the space $\mathbf{X}$. This approach is justified due to the near-linearity of the sigmoid function and the not too excessive length of the two set of vectors. In general the input space $\mathbf{X}$ (as viewed from the hidden space $\mathbf{H}$) can be visualized as a finite hyperplane with its edges terminating abruptly in space due to the finite asymptotic values of the sigmoid function. For hidden vectors of not too excessive length, its projection will fall entirely inside the finite hyperplane and will not have its tips projected beyond the edges. Under this circumstances the linearity assumption can be justified to a high degree, and the relationship between the two spaces can be adequately described by the above diagram.

The above argument is sound except for our assumption that we have a priori knowledge about the hidden space **H** which best represents the T-vector space $\mathbf{D}^{-1}$. In fact, finding this optimal hidden space **H** has been our chief motivation in setting up the above model in clarifying the knowledge representation strategy of neural networks, so it seems that we are running into a cyclic argument. However, there is one situation in which we do have some prior knowledge concerning the hidden T-vectors, and this is the situation when the neural network is within a dynamic node creation environment, in which an estimation for a new hidden T-vector can be obtained as a function of the target T-vectors $\mathbf{d}_k$ and the old hidden T-vectors. It will be seen in the next subsection that the above model concerning the relationship between the input and hidden space fits neatly into the framework of a dynamic node creation environment.

5.6 An Inspiration for a New Training Algorithm from the Current Model

The knowledge representation model described in the previous subsections directly leads to a viable implementation of a practical dynamic node creation scheme. Usually, the most important part of a node creation scheme involves the design of the hidden weights and output weights for the new hidden node. For example, both the designs of the progressive training scheme [9] and the growth algorithm described in Chapter 4 place great emphasis on the design of the new hidden weights and the new output weights as their specifications not only ensure the optimality of the new node with respect to the old network but actually dictates the ultimate convergency of the corresponding training scheme. However, as has been mentioned before, their hidden node initialization schemes which depend on only a single training data will most possibly result in the memorization of noisy training patterns. It will be seen later that the model derived above allows the possibility of initializing the new hidden node with multiple training patterns.

The only occasion in a neural network when we have some a priori knowledge concerning the hidden T-vectors $\mathbf{h}_j$ is when we are under a dynamic node creation environment. Under this context we can derive an estimation for the new hidden T-vector $\mathbf{h}_Q$ as a function of T-vectors in the $\mathbf{D}^{-1}$ space and the old hidden T-vectors $\mathbf{h}_1,\ldots\ldots, \mathbf{h}_{Q-1}$, provided that we assume an adequate representation of the training set by the appended neural network after the addition of the current new node, which is not an unreasonable assumption as we should take every opportunities to minimize the number of hidden nodes in a neural network: if the training set can really be adequately represented by the addition of a single node, and we initialize the weights of the new hidden node in such a way as to be compatible with this expectation, we can actually solve our problem using the minimum number of additional nodes, while any deviation from the above initialization may lead to unnecessary addition of hidden nodes.

It is appropriate for us to give a brief sketch of the training algorithm inspired from the above model, the full algorithm presentation will be given in the next chapter. To summarize from the above analysis, for a hidden T-vector $\hat{\mathbf{h}}_Q$ to be considered as a valid candidate for adequately representing a training set, it must satisfy the following two criteria:

(1) It must be close enough to the T-vector space $\mathbf{D}^{-1}$.

(2) It must be close enough to the input space $\mathbf{X}$.

For simplicity's sake, we first consider the single-output network. In other words, the T-vector space $\mathbf{D}^{-1}$ contains only the vector $\mathbf{d}_1^{-1}$. According to the above discussion, the estimation for the new hidden T-vector $\beta_Q\mathbf{h}_Q$, which we will call the target T-vector (the presence of the factor β_Q will be explained in the next chapter. We can temporarily regard the above combination as a single entity) can be expressed as a linear combination of $\mathbf{d}_1^{-1}$ and the old hidden T-vectors $\mathbf{h}_1,\ldots\ldots,\mathbf{h}_{Q-1}$

$$\beta_Q\hat{\boldsymbol{h}}_Q=\boldsymbol{d}_1^{-1}+\sum_{j=1}^{Q-1}\beta_j\boldsymbol{h}_j \tag{5.4}$$

This is the most general expression for the estimation of the new hidden T-vector. By the above expression, we have assumed that $\beta_Q\hat{\mathbf{h}}_Q$ is within the space $\mathbf{D}^{-1}$.

The next task concerns the selection of the coefficients $\beta_1^*,\ldots,\beta_{Q-1}^*$ such that the resulting estimation is closest to the input space **X**. Mathematically speaking, we would like to minimize $\|\beta_Q\hat{\mathbf{h}}_Q-\mathbf{P}_X\beta_Q\hat{\mathbf{h}}_Q\|^2$ where $\mathbf{P}_X$ is the projection operator on **X**. The resulting $\beta_Q\hat{\mathbf{h}}_Q^*$ can be solved by linear optimization method and results in an estimation which is closest to **X**. The situation is depicted in the following diagram:

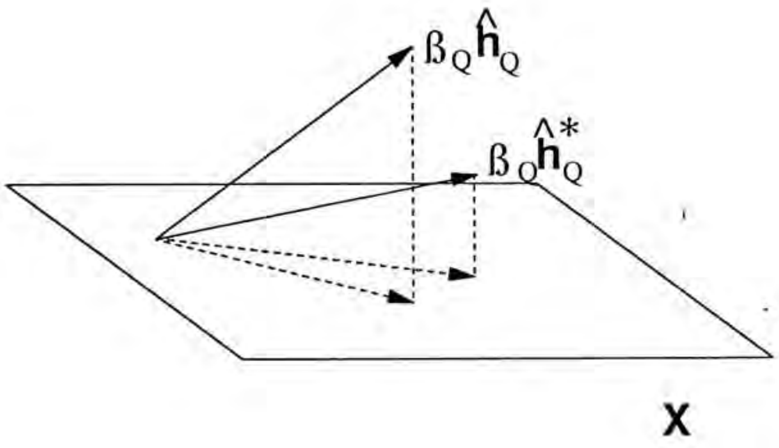

Fig 5.4 The Optimization of $\beta_Q\hat{\mathbf{h}}_Q$

From the optimum target vector $\beta_Q\hat{\mathbf{h}}_Q^*$, we can obtain the true hidden T-vector $\mathbf{h}_Q$ as described in Chapter 6. The true T-vector $\mathbf{h}_Q$ may neither be in the T-vector space $\mathbf{D}^{-1}$ or in the input space **X** due to the presence of the sigmoid function. But the above procedures guarantee sufficient closeness of this resulting $\mathbf{h}_Q$ to either the hidden space **H** and the input space **X** such that it qualifies as a candidate T-hidden vector in a neural network which efficiently represents the training set.

The situation is slightly more complicated for the multi-output neural network. For this case the T-vector space $\mathbf{D}^{-1}$ is spanned by the T-vectors $\mathbf{d}_1^{-1},\ldots,\mathbf{d}_m^{-1}$. Though we can apply the above procedure to each of the $\mathbf{d}_k^{-1}$, the original set of T-vectors is not the only set of basis which spans the T-vector space $\mathbf{D}^{-1}$. Unlike the case for a single output in which we have no further choice in selecting our basis, we can transform the T-vectors $\mathbf{d}_k^{-1}$ into another set of basis vectors $\mathbf{c}_k^{-1}$ which span the same space $\mathbf{D}^{-1}$ but with their tips being much closer to the input space **X**. After this, we apply the single-output optimization procedure to each of the T-vectors $\mathbf{c}_k^{-1}$. Since each of these T-vectors is now much closer to the input space **X**, these vectors will be much more well represented by T-vectors in **X** than the original T-vectors $\mathbf{d}_k^{-1}$. In addition, the number of hidden nodes required to approximate each $\mathbf{c}_k^{-1}$ will be reduced and thus results in an overall reduction in network size. Finally, since these vectors $\mathbf{c}_k^{-1}$'s span the same space $\mathbf{D}^{-1}$, our purpose of exact training set representation can be equivalently achieved. This situation is depicted in Fig.5.5.

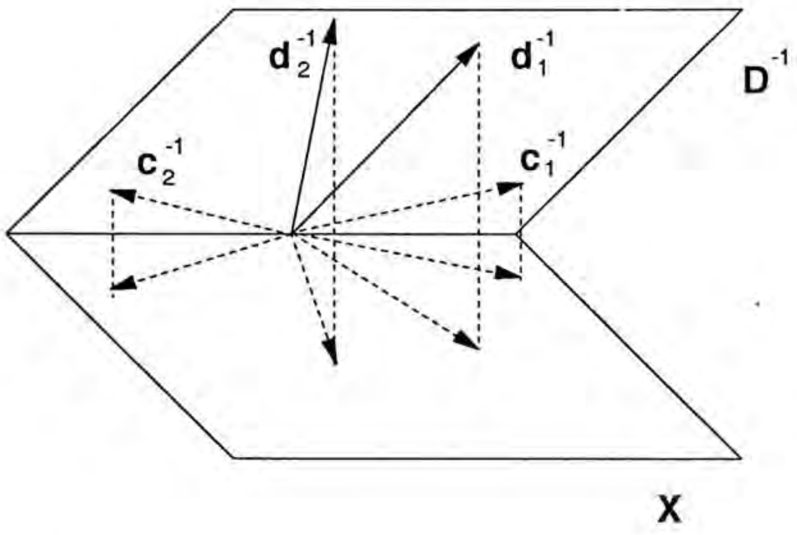

Fig 5.5 Transformation of the Basis Spanning $\mathbf{D}^{-1}$

However, in carrying out the above procedure, we may have to tackle the following issues:

(1) We must rotate the $\mathbf{c}_k^{-1}$'s such that their tips should be as close to the input space **X** as possible, while keeping them in the same space $\mathbf{D}^{-1}$.

(2) The transformed $\mathbf{c}_k^{-1}$'s should still be linearly independent from each other such that the dimension of $\mathbf{D}^{-1}$ is maintained.

We will addressed these issues altogether in Chapter 6 in the form of the multi-output version for the deterministic training algorithm.

6. THE DETERMINISTIC TRAINING ALGORITHM FOR NEURAL NETWORKS

6.1 Introduction

From the knowledge representation model of the previous chapter, it is clear that we should derive a neural network training scheme such that it conforms with the knowledge representation model mentioned in the previous chapter. In this way, the algorithm will eventually lead to a viable solution for the training set at hand. As has been mentioned the model involves the input space **X**, the hidden space **H** and the inverse desired output space $\mathbf{D}^{-1}$. The primary objective of the new algorithm should be such that a proper relationship is maintained between these three spaces.

In addition, the new algorithm must address some of the problems which constantly plagues the classical BP algorithm such as the local minima problem, the indeterminate architecture problem and the generalization problem, etc. It has been mentioned that the progressive training algorithm [9] has addressed both the problem of local minima and the problem of indeterminate architecture, but the algorithm has brought up other problems which are equally pressing and which requires a new perspective on data representation in neural network such that the new algorithm resulting from the new perspective not only solves the present problems of progressive training (and indirectly those of classical BP), but may lead to new insights in future courses of neural network training algorithm development.

A brief recapitulation on our course of research is appropriate here: we have mentioned that one of the defects of progressive training lies on the dependence of the parameters of the new hidden node on a single training data. Whenever the training data is contaminated with a non-negligible amount of noise, and when incidentally this training data is selected as the candidate for the initialization of a new hidden node, (which is most likely due to the often large magnitude deviation of a noisy training data from the rest of the training set, and from the viewpoint of the progressive training algorithm, this signals the requirement of a new hidden node to cater for the noisy training data) the memorization of that noisy training data which is alien to the training set would result.

It has been mentioned that the progressive training algorithm has other problems, such as its artificial training sequence which results in the more frequent exposure of the training data with small Euclidean norms to the network and the proportional reduction in exposure frequency to the network for those training data with large Euclidean norm. This training sequence eventually results in an equivalent training sequence in which the training data with small Euclidean norm occurs will higher probability than those training data with large Euclidean norm. This training scheme may eventually lead to a biased solution of the problem at hand where the phenomenon of over-fitting may be prominent in those regions around the origin in the function domain, and where the problem of under-representation may occasionally be observed at the periphery of the support region of the mapping, thus resulting in an inaccurate representation of the mapping by the neural network when compared to those networks trained with a standard training set with each training data occurring with equal probabilities.

It is seen that the growth algorithm introduced in the previous chapter has solved the above problem by adopting an additional input node for each hidden node, thus converting the original sigmoid node into a quasi-radial basis network which can easily isolate any training data from the rest of the training set by choosing that training data as the centre of the new hidden node. Moreover, the algorithm is formulated in such a framework such that conventional BP algorithm can be directly applied to the network without further modification (for conventional RBF network, separate centre and variance adaptation equations have to be used). However, the growth algorithm has not solved the first problem mentioned at the start of the chapter: that the new hidden node initialization scheme is still based on a single training data only. As a result, the chief merit of the growth algorithm only lies in its ability to cater for an unordered training set as compared to the progressive training algorithm in which an ordered training set is crucial to the convergence of the algorithm. A new hidden node initialization scheme has yet to be found in which the hidden weights and output weight values are based on multiple training data. In this way, each hidden node on the average accounts for more that one training data and is thus biased against the memorization of a single training data. Moreover, this arrangement ensures that the number of hidden nodes will be well below the number of training data.

Thus, we are facing a dilemma here: on the one hand, we have to ensure that each new hidden node should cater for more that one training data such that the number of hidden nodes would be well below the training set size and the overall generalization capability of the network would not be seriously disturbed by the memorization of a single noisy training data by a new hidden node. On the other hand, the single data initialization requirement is the very condition which ensures the convergence of these two previous algorithms. In other words, a new hidden node initialization strategy has to be found which simultaneously allows multiple training data initialization and ensures the convergence of the resulting network configuration to the desired solution. The new model introduced in the previous chapter provides us with the solution: the key to the convergence of the network lies in the linear independency of the hidden vectors.

6.2 The Linear Independency Requirement for the Hidden T-vectors

With reference to the previous chapter, we devote the space spanned by the hidden T-vectors as the hidden space **H**, the space spanned by the inverse desired output T-vectors as the inverse desired output space $\mathbf{D}^{-1}$, and that spanned by the input T-vectors as the input space **X**. From this new perspective on the knowledge representation mode in neural networks we can immediately see that a necessary and sufficient condition for the adequate representation of a training set by the neural network is the inclusion of the inverse desired output space $\mathbf{D}^{-1}$ in the hidden space **H**.

By the very word "span" we have assumed the linear independency of the basis vectors in the corresponding space. However, this condition is not ensured for most real world training data set. For example, either the input or desired output space or both may include linearly dependent T-vectors.

The linear independency condition of the input T-vectors and output T-vectors can be checked by the Gram-Schmidt orthogonalization procedure, and the linearly dependent T-vectors can thus be duly removed such that the remaining T-vectors truly span the input or output space. As a result, the adherence of the input and output T-vectors to the condition of linear independency is directly under our control.

The situation becomes more complicated if we consider the linear independency of the hidden T-vectors, as

(a) We cannot predetermine the values of each component of the hidden T-vectors, and

(b) We cannot predetermine the eventual number of hidden T-vectors (or number of hidden nodes when viewed in the network perspective) required to accurately represent the training set.

As can be seen from these two problems, we cannot be satisfied with only the existence of a method for checking the linear independency of the hidden T-vectors, we must also alter the hidden T-vectors appropriately such as to ensure their linear independency. As has been mentioned in the Introduction section, the new deterministic algorithm will still include a hidden node addition mechanism in which the initial weights and bias will depend on multiple training data. In other words, besides building a measure of optimality into the new hidden node initialization strategy , as in the case of the progressive training algorithm and the growth algorithm, we must also further alter these weights and biases in such a way that the new hidden T-vector is linearly independent from the hidden T-vectors generated by the old network.

It will be seen later in this chapter that the linear independency condition can be achieved by progressively reducing the size of the training subset used to initialize the new hidden node. It is proved that whenever the size of the subset is reduced beyond the number of hidden nodes, the new hidden T-vector can be made to be linearly independent from the old hidden T-vectors and thus at some point in the course of reducing the training subset we are guaranteed to obtain a linearly independent hidden T-vector. The progressive training algorithm and growth algorithm are seen as limiting cases of the present algorithm in which the training subset is stripped of all but one training data (the present algorithm will never reach this stage due to the reason above), and this training data is utilized to construct an linearly independent hidden T-vector (an obvious intention of these two algorithms though the corresponding initialization schemes are phrased in different wordings.) In light of this, the present algorithm can be viewed as a generalization of these two algorithms with the added advantage that no new hidden node will ever be initialized by a single training data.

At this point one may appreciate the significance of the linearly independency requirement: whether or not a new hidden T-vector is linearly independent from the old hidden T-vectors signifies whether the new hidden T-vector contains any new information : if the new hidden T-vector is linearly dependent on the old hidden vectors, it means that any information contained in the new hidden vector is embedded in the old hidden vectors. In other words, the new hidden T-vector does not bring any new information to the network. This directly affects the convergence of any node addition algorithm as the exclusion of any new information signifies a non-decreasing mean square error which is synonymous with non-convergence.

6.3 Inspiration of the Current Work from the Barmann T-Vector Model

The algorithm proposed by Barmann et al [2] adopts a new iterative learning procedure for a single-layer artificial neural network which is different from the conventional BP approach. In summary, the algorithm changes the hidden node outputs of all the hidden nodes in the network such that the "solvability condition " is satisfied: when phrased in the terminologies of Chapter 5, the "solvability condition " is equivalent to saying that all the inverse desired output T-vectors $\mathbf{d}_k^{-1}$ are within the hidden space. The authors ensured this condition by cyclically adapting the hidden weights of each hidden node in turn in such a way that the solvability condition is eventually satisfied. Suppose that the current network contains q hidden nodes and we are currently adapting the hidden weights of the r-th hidden node. The current error T-vector $\mathbf{E}_j$ (assuming a single-output network) is defined by Eq (6.1):

$$E_j=d_1^{-1}-\sum_{j=1,j\neq r}^{q} u_{jl}h_j \tag{6.1}$$

where u_{j1} represent the output weights and $\mathbf{h}_j$ signifies all the hidden T-vectors excluding the r-th hidden T-vector. An attempt is then made to maximize the following "normalized scalar product" S_j between the error T-vector $\mathbf{E}_j$ and the j-th hidden T-vector $\mathbf{h}_j$

$$S_j=\frac{<E_j,h_j>^2}{\|h_j\|^2} \tag{6.2}$$

The maximization of this function is achieved through a gradient ascent procedure in which the adaptation equation for the various hidden weight is obtained by differentiating S_j with respect to all the hidden weights. After the completion of this maximization procedure, the output weight of the new node is obtained by projecting the error T-vector $\mathbf{E}_j$ onto $\mathbf{h}_j$

$$u_{jl}=\frac{<E_j,h_j>}{\|h_j\|^2} \tag{6.3}$$

The procedure is divided into 2 procedures: in the first part , a number of inner iterations C_i is performed for each hidden node which correspond to the maximization of S_j for each hidden node and the subsequent determination of the output weight u_{j1}. A sweep through all the hidden nodes of the network is known as an outer iteration. The completion of an outer iteration is typically followed by several "post-iterations" which continuously apply Eq (6.2) to all the output weights of the network. This latter procedure is adopted since it usually takes a longer time for the algorithm to complete an inner iteration while the time spent on performing a "post-iteration" is much shorter. Therefore, it is worthwhile to spend more time on the "post-iteration" stage such that the accuracy of the output weights are increased before applying the inner iteration. The flow of the algorithm is depicted in the flowchart in fig 6.1.

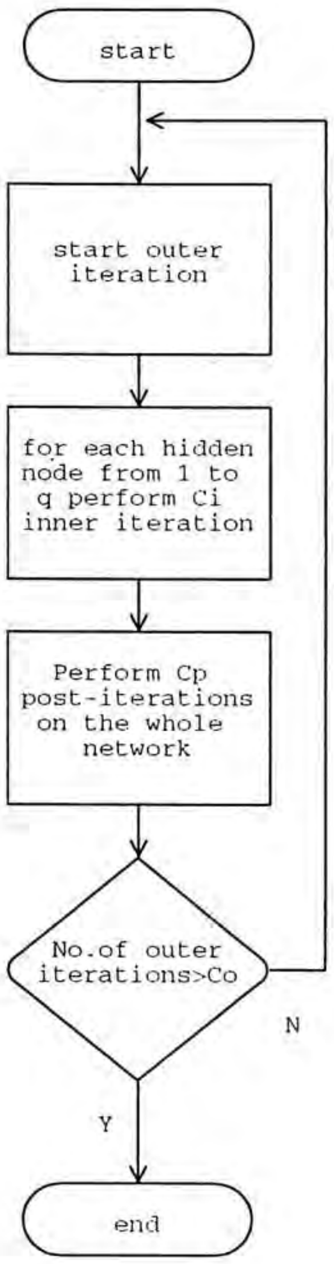
start
start outer iteration
for each hidden node from 1 to q perform Ci inner iteration
Perform Cp post-iterations on the whole network
No.of outer iterations>Co
N
Y
end

Fig 6.1 Flow of the Algorithm proposed by Barmann et al

In the flowchart, the symbol C_i indicates the number of inner iterations, C_o indicates the number of outer iterations, and C_p indicates the number of post-iterations.

For the multi-output network, we have to select a T-vector $\mathbf{E}_j$ in order to carry out the inner iterations. If the network has m output nodes, we would in general have m such error vectors which are indexed by the double subscript $\mathbf{E}_{jk}$, k=1 to m. Barmann proposed that we should select the $\mathbf{E}_{jk}$ with the greatest norm, and use this T-vector to carry out the inner iteration step. From the above discussion it is seen that, though the authors are not explicitly developing a dynamic node creation algorithm, the above algorithm can be easily adapted to include a node addition procedure as the hidden nodes are adapted cyclically in the inner iteration process.

It seems that we can directly modify the above algorithm into a dynamic node creation algorithm and adopt the iterative learning procedure instead of the conventional BP process. However, it is noticed that, when we apply the above algorithm to the parity problem from order 2 to 4, the convergence failure rate of the network is very high. The situation becomes more and more serious as the order of parity increases. In Table 6.1, we have tabulated the results of our simulations together with the conventional BP results to serve as a comparison . The various parameters are defined as C_i=4, C_p=5

Parity Order	No. of Outer Iterations (Barmann's Algorithm)	No. of Epochs (BP Algorithm)
2	1(80%)	107(0%)
3	-- (100%)	62(0%)
4	-- (100%)	698(80%)

Table 6.1 Comparison between Barmann's Algorithm and the BP Algorithm

The figure in the parentheses indicates the failure rate among 10 trials of each algorithm. Since a single outer iteration corresponds to approximately 7 epochs of BP as estimated by the authors, it is seen that the convergence rate of Barmann's algorithm is extremely fast if the number of training patterns is approximately the same as the number of nodes as in the case for the parity-2 problem. However, it is observed that the failure rate of the new algorithm is also very high. This situation was further worsened when the algorithm is applied to higher order parity problem where none of the trials converged. The high failure rate of the above algorithm may be due to the following reason:

In the cost function for the maximization of S, the expression involves the factor $\mathbf{E}_j$ which in turn depends on only the inverse desired output T-vector $\mathbf{d}_k^{-1}$. In other words, the cost function does not take into consideration the function provided by the nonlinearity at the output node. Suppose that $\mathbf{y}_k^{-1}$ is the output of the network prior to the output nonlinearity, the maximization of S would have the effect of minimizing the sum squared error between $\mathbf{y}_k^{-1}$ and $\mathbf{d}_k^{-1}$. However, this process only minimizes the overall sum-squared error and does not consider the distribution of error at the various training patterns. Considering a single component $y_k^{-1}(t)$ of $\mathbf{y}_k^{-1}$ and a single component $d_k^{-1}(t)$ of $\mathbf{d}_k^{-1}$. Through the above optimization process, we would like to minimize the difference between $y_k^{-1}(t)$ and $d_k^{-1}(t)$. However, if $d_k^{-1}(t)$ and $y_k^{-1}(t)$ both have large magnitudes, such that they are within the saturation region of the output nonlinearity, the compression effect of the output nonlinearity will bring them close together even though they may have a large difference. In other words, the difference between the two components is less important when their magnitudes are large than when they are small. The current optimization of S does not take this fact into account, and thus the optimization process would attempt to minimize the error at those patterns where the weighting of the error is not so important at the expense of those patterns where the minimization of the error is critical,thus resulting in an overall mapping which is less desirable than one which is obtained using a conventional BP network.

In view of these , we should adopt the BP process for the optimization in the above algorithm if we are to modify the above algorithm into a dynamic node creation algorithm. The maximization of S should only be considered as an approximate process which should then be followed by the global BP process. However, this practice would be excessively cumbersome since it involves a double iterative learning procedure in which the result of the first optimization process would soon be rendered invalid by the global BP optimization process. As a result, it would be ideal if we can deterministically assign an initial state for the new hidden node such that the maximization of S is approximately satisfied and then apply the BP fine-tuning process to the network. The initialization scheme of the deterministic training algorithm described in this Chapter is developed along this line with the purpose of a similar optimization criterion: that of the distance between the target vector $\beta_Q \hat{\mathbf{h}}_Q$ and the input space **X**. The definition of this target vector will be given later on. However, we would first describe the general framework of a dynamic node creation algorithm and how the various steps of the general framework correspond to the steps in the present deterministic algorithm.

6.4 General Framework of Dynamic Node Creation Algorithm

From the description of the progressive training algorithm and the growth algorithm we can generalize their methodologies into a general framework for a dynamic node creation algorithm which is given in the flowchart in fig 6.2:

The section number beside some boxes identify those subsections which describe the corresponding procedure in the present deterministic algorithm. The meaning of the procedure described in most of the boxes is self-explanatory, except for the following terms:

(1) Complete Convergence, which means that the output error of the network has fallen below a pre-specified threshold.

(2) Partial Convergence, which means that a temporary convergence criterion is achieved and which differs from algorithm to algorithm. For example, in the progressive training scheme it means the complete convergence of the network with respect to the partial training set. In the growth algorithm, it means that a portion of the training patterns which is equal to the present number of hidden nodes is within a certain error threshold. In the deterministic algorithm this implies the linear independency of all the hidden T-vectors in the restricted network.

(3) The tuning of the network parameters: for any dynamic node construction scheme, one must first build up an artificial model for the new hidden node which will ensure the partial

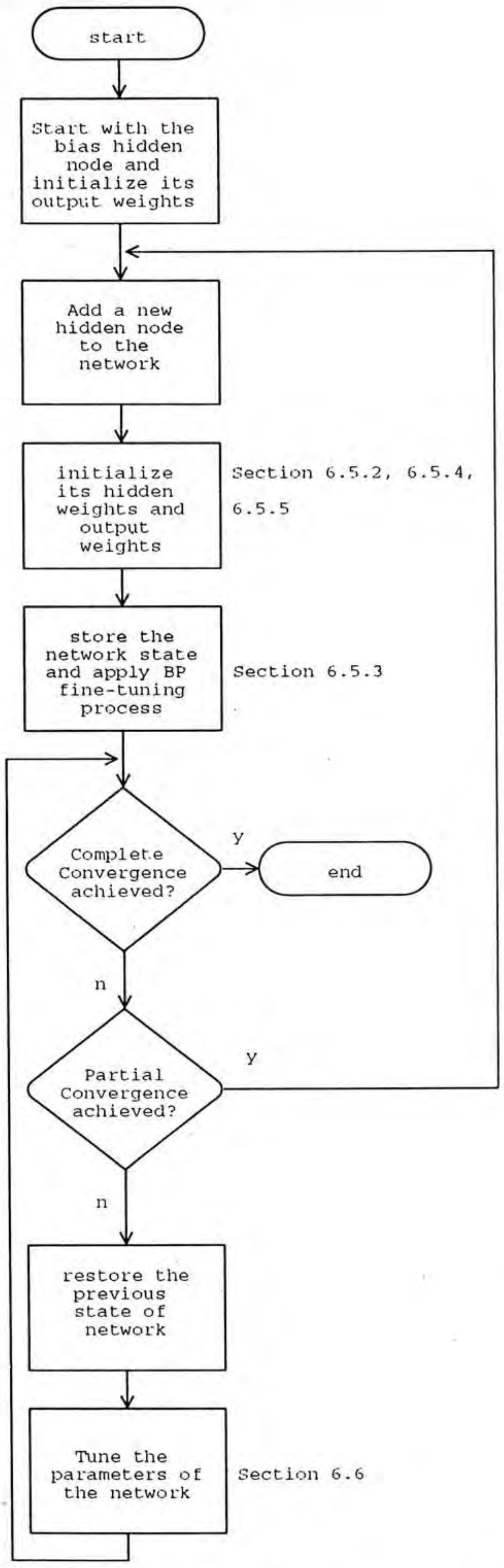
start
Start with the bias hidden node and initialize its output weights
Add a new hidden node to the network
initialize its hidden weights and output weights
Section 6.5.2, 6.5.4, 6.5.5
store the network state and apply BP fine-tuning process
Section 6.5.3
Complete Convergence achieved?
y
end
n
Partial Convergence achieved?
y
n
restore the previous state of network
Tune the parameters of the network
Section 6.6

Fig 6.2 The general framework for a dynamic node creation algorithm

convergence of the appended network and then relax this constraint and apply the BP algorithm to the network in the hope of finding a better overall solution. For example, in the progressive training algorithm this model is a step function with the new training pattern situated at the step transition. In the growth algorithm the model is an impulse which peaks at the "difficult" pattern. It will be seen in subsection 6.6 (as indicated in fig 6.2) that we have prepare a corresponding model in the present algorithm for the new node which ensures its linear independency from the old hidden T-vectors. However, when the node is first added to the network , this model is relaxed in the form of the smooth sigmoid transition in the progressive training algorithm and the smooth quasi-radial basis function in the growth algorithm such that the BP process can operate in a normal network environment. However, the final exact model is approachable through the gradual scale-up process in both of the above algorithms if the network fails to achieve partial convergence through the BP process. The corresponding tuning process for the present algorithm is described in subsection 6.6.

In summary, it could be seen that the present algorithm fits the overall framework of a dynamic node algorithm and the corresponding process is described in the indicated subsection: the new hidden node initialization scheme is described in subsection 6.5. In subsection 6.5.3 we describe some pre-processing steps to the parameters of the new hidden node before applying the BP fine-tuning process such that the iterative learning procedure can be operated in a more favourable pedestal. In subsection 6.6, we describe the partial convergence assurance procedure which guarantees the eventual linear independency of the new hidden T-vector from the old hidden T-vectors.

6.5 The Deterministic Initialization Scheme for the New Hidden Nodes

6.5.1 Introduction

The new node initialization scheme of the present algorithm consists of the following modules as expressed in the flowchart in fig 6.3

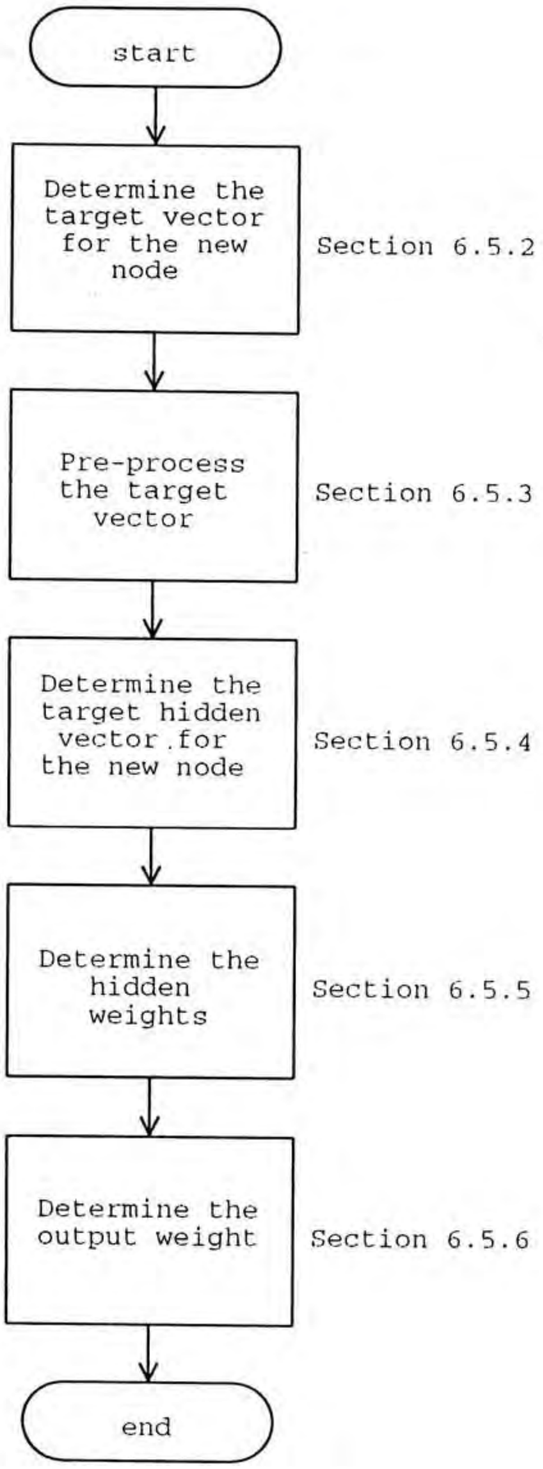
start
Determine the target vector for the new node
Section 6.5.2
Pre-process the target vector
Section 6.5.3
Determine the target hidden vector for the new node
Section 6.5.4
Determine the hidden weights
Section 6.5.5
Determine the output weight
Section 6.5.6
end

Fig 6.3 Flow of the hidden node initialization scheme

The description of the process in each box is contained in the subsection number beside the box. A synopsis of the contents of each subsection is given below:

In subsection 6.5.2, we determine the target T-vector symbolized by $\beta_Q\hat{\mathbf{h}}_Q$ for the new hidden node (the meaning of the symbols and the multiplicative factor β_Q will be given in the subsection). The target T-vector is distinguished from the target hidden T-vector $\hat{\mathbf{h}}_Q$ (to be mentioned shortly) by the multiplicative factor β_Q. As a result these two T-vectors point in the same direction. The purpose of the determination of these two T-vectors is to provide a guideline for calculating the hidden weights of the new hidden node. An important notion to be introduced in this subsection is the near linearity assumption of the sigmoid nonlinearity which allows a simple criterion for the selection of the target T-vector to be applied to the node initialization process: that the distance of the target vector (which is generated from the inverse desired output space $\mathbf{D}^{-1}$ and the old hidden space $\mathbf{H}$) should be as near to the input space $\mathbf{X}$ as possible so that the resulting target hidden vector (derived from the target vector) could be well approximated by the input space $\mathbf{X}$.

In subsection 6.5.3, a pre-processing step for the target vector is introduced. In general, the target T-vector is generated in a deterministic way from the inverse desired output space $\mathbf{D}^{-1}$ and the old hidden space $\mathbf{H}$, and there is no guarantee that the subsequent BP fine-tuning process will find itself situated in a favourable position on the error surface such that it can begin the descent process efficiently. In this subsection, guidelines will be given for deriving a suitable pre-processing step for the target vector such that the above purpose is achieved.

In subsection 6.5.4, we would describe our methodology for determining the target hidden T-vector from the target T-vector. As we have seen, since these two T-vectors are pointing in the same direction, what we have to do is to perform a simple scaling on the target T-vector. The scaling factor should be chosen such that the magnitude of the resulting target hidden T-vector is small enough in order that the near linearity assumption of the sigmoid nonlinearity is satisfied.

In subsection 6.5.5, the actual process of node initialization begins by the determination of the hidden weights of the new node from the target hidden vector which is all the information that are required.

In subsection 6.5.6, the process of determining the output weights of the appended network would be introduced, the determination process would involve the T-vectors in the inverse desired output space $\mathbf{D}^{-1}$ and all the hidden T-vectors in the network.

6.5.2 Determination of the Target T-vector

6.5.2.1 Introduction

Based on this notion, a target vector for the new node would be defined which would be instrumental in generating the hidden weights of the new node. The term "target" is adopted as we would like the actual hidden T-vector of the new node to actually resemble this target T-vector.

We could summarize the generation process of the target vector in the following two steps:

(1) Formulate a model for the target vector in terms of the T-vectors in the inverse desired output space $\mathbf{D}^{-1}$ and the old hidden T-vectors $\mathbf{h}_1,..., \mathbf{h}_{Q-1}$ of the old network.

(2) Determine the parameters of the model such that the resulting target vector is closest to the input space $\mathbf{X}$.

For step (1), the formulation of the model of the target T-vector would depend on an assumption which is to be introduced in the subsection 6.5.2.2. The realization of step (2) depends strongly on the near linearity assumption of the sigmoid nonlinearity which will be discussed in subsection 6.5.2.3. Finally, the evaluation for the target T-vector would also be described in subsection 6.5.2.3.

The deterministic initialization scheme for the new hidden nodes will now be introduced: this introduction will draw heavily on materials from the Chapter 5 as the present algorithm is wholly inspired by the data representation model described there.

As described before, the present algorithm incorporates a dynamic network architecture in order to counteract the problem of indeterminate architecture. As a result, the focus of the new algorithm should be on the initialization scheme of the new hidden node as in the previous two algorithms. The new algorithm differs from the previous two algorithm in that multiple training data are used to initialize the new hidden node such that memorization of a noisy data by the new node is avoided. Moreover, no special assumption or segregation of the training set (as in progressive training) or of the network structure (as in the growth algorithm) is required. In other words, the input to the algorithm is the full-sized training set and the output of the algorithm is a standard BP network.

Let us first review our concept of input space **X**, hidden space **H** and inverse desired output space $\mathbf{D}^{-1}$. For simplicity it has been mentioned that the neural network can perfectly represent the training set if the inverse desired output space $\mathbf{D}^{-1}$ is included in the hidden space **H**. It is also known that the hidden space is derived from the input space **X** through the nonlinear sigmoid transformation. Without the sigmoid transformation, no enhancement to the solution is possible through multiple hidden nodes, and the network can as well be represented by a single linear node. However, the sigmoid transformation is not a drastically nonlinear transformation in that a large portion of the function approximates a linear function. Due to this nature, all hidden T-vectors should be reasonably "close" to the input space **X** in order to be well approximated by the input T-vectors. In other words, the new node initialization scheme must choose among all the possible new hidden T-vectors the one which is "closest" to the input space **X**, such that the former can be well approximated by the latter.

6.5.2.2 Modelling of the Target Vector $\beta_Q \hat{\mathbf{h}}_Q$

The modelling of the target vector $\beta_Q \hat{\mathbf{h}}_Q$ depends on the assumption that on addition of the next new hidden node, the problem will be fully solved. If this is assumed and the training set can really be adequately represented on the addition of the next new hidden node, then we have made a decision compatible with this reality and the final network will contain the minimum number of nodes required to solve the problem. Otherwise, there is the possibility that additional nodes would be required before the network converges and thus the network would contain redundant nodes. Following this assumption and assuming a single-output network, we model our expected new hidden vector as

$$\beta_Q \hat{h}_Q = d_1^{-1} + \sum_{j=1}^{Q-1} \beta_j \hat{h}_j \qquad (6.4)$$

The T-vector $\beta_Q\hat{\mathbf{h}}_Q$ would be what we call the target vector. A slight rearrangement of Eq (6.4) would clarify the adoption of this equation as our model as shown in Eq (6.5)

$$d_1^{-1} = -\sum_{j=1}^{Q-1} \beta_j h_j + \beta_Q \hat{h}_Q \qquad (6.5)$$

which is equivalent to saying that the desired output of all the training patterns can be approximated by a neural network with output weights $\beta_1,.....,\beta_Q$ and hidden T-vectors $\mathbf{h}_1,.....,\mathbf{h}_{Q-1},\hat{\mathbf{h}}_Q$. It is seen that for every $\beta_1,\beta_2,.....,\beta_{Q-1}$, the corresponding target T-vector $\beta_Q\hat{\mathbf{h}}_Q$, when considered as a new hidden T-vector, and added to the network, would allow the exact representation of $\mathbf{d}_1^{-1}$. Therefore, we would adjust the coefficients $\beta_1,.....,\beta_{Q-1}$ such that the resulting target T-vector is closest to the input space **X**, so that the actual hidden T-vector emitted by the new hidden node would somewhat resemble this target T-vector, since all hidden T-vectors eventually originate from the input space **X**. Originally, the definition of a distance measure between the target T-vector and the input space **X** is invalid due to the presence of the sigmoid nonlinearity in the hidden node. However, if some portion of the nonlinearity resembles a straight line, we can still talk about the distance between the target T-vector and the input space **X** by suitably restricting the magnitude of the target T-vector which is equivalent to controlling the magnitude of the target hidden T-vector through the parameter β_Q. The requirement of near linearity condition of the sigmoid function will be given in the next subsection.

6.5.2.3 Near-Linearity Condition for the Sigmoid Function

The near-linearity assumption of the sigmoid function is derived such that the double iterative learning procedure associated with the derivation of a dynamic node construction scheme through the T-vector approach can be avoided as mentioned in the section 6.1 of this Chapter. The double iterative learning procedure consists of a global BP process which tunes the parameters of the whole network and a local update process which tunes the parameters of the new hidden node. Normally, researchers, when designing neural network according to the T-vector model, adopt only the latter approach. For example ,the cascade correlation algorithms [12] and the algorithm proposed by Barmann et al [2], utilizes an approach in which the hidden weights w_{iQ} of the new node are iteratively adjusted to align the resulting hidden T-vector to the residual vector. However, they have utilized this local adaptation approach as the exclusive optimization procedure for their networks in the form of freezing the other weights in the old network while adapting the new node as in cascade correlation algorithm or adapting each hidden node in turn as in Barmann et.al [2].

It has been mentioned in the previous chapters that the approach of the first algorithm, by freezing the weights of the old network, severely restricts the degrees of freedom of the neural network. For the second approach, it has been mentioned that the cost function of the local adaptation involves only the space $\mathbf{D}^{-1}$ and $\mathbf{H}$, and does not take into consideration the output sigmoid nonlinearity. As a result, the final solution may exhibit a distribution of errors among the various training patterns which is incompatible with the sigmoid function. In view of these, the BP process would serve well as the iterative optimization procedure after node initialization since:

(1) It is simple to implement.

(2) It provides a global update for all the weights in the network.

(3) It takes into account the characteristics of the output nonlinear function.

Therefore, the ideal algorithm should incorporate a hybrid process which includes both the BP global update procedure and a local hidden node update procedure, but this hybrid process involves a repetition of gradient descent processes in which the first local update is based on those information (the frozen weights and bias of the old network) which will be rendered invalid soon after by the global BP update process. As a result, excessive accuracy in the first process is not necessary and the result of this local update is at best transient. Instead, it is much more desirable if we can synthesize a new hidden T-vector for the hidden node by deterministic methods (rather than by iterative method) which is reasonably close to the input space **X**, and then immediately apply the global BP process to the network. In this way we can avoid the repetition of the gradient descent processes. To achieve this, we have to make an approximation on the sigmoid function. As have been mentioned, the central portion of the sigmoid function closely approximates a linear function. To avoid the iterative adaptation approach, we can treat the sigmoid function simply as a linear function, and the previous iterative procedure would reduce to a deterministic weight assignment procedure. In this way the double gradient descent procedure mentioned previously has been avoided. The linear approximation is valid whenever the magnitude of each component of the new hidden vector remains within the linear range of the sigmoid function, which extends from approximately the sigmoid function value f(x) of 0.2 to 0.8. This can be controlled by appropriately increasing the multiplicative factor in the target T-vector $\beta_Q\hat{\mathbf{h}}_Q$ such that all the components of $\hat{\mathbf{h}}_Q$ are within this magnitude. This practice in turn implies small hidden weight magnitudes for the new hidden node. The above linear approximation does not actually pose such a serious disadvantage and accuracy compromise to the hidden vector optimization process as the procedure seems to suggest as

(1) the deviation of the central portion of the sigmoid function from linearity is actually very small, even within the above rather large functional range of 0.2 to 0.8, which includes most instances of hidden node output. Therefore, the above assumption serves as an appropriate model for the actual hidden node.

(2) Small hidden weights for the network actually facilitates generalization [11] as the resulting sigmoid transition is less abrupt and the overall functional shape applies to a larger domain. As a result, we should at least allow the new hidden node to pass through this stage prior to further training instead of clamping it to the highly

saturated extremes of the sigmoid function and

(3) We would further apply the BP process to fine-tune the network and thus any local hidden node training process needs only be approximate. Besides, our initialization process places the extreme values of the new hidden vector components at 0.2 and 0.8, which is almost at the onset of the nonlinearity. Thus any requirement of a saturated hidden node (e.g. in memorizing any peculiar pattern in the training set) would be quickly met through the BP fine-tuning process.

We first based our discussion on neural networks with single output. We will next extend our algorithm to neural networks with multiple outputs in subsection 6.7. As mentioned before, the target T-vector $\beta_Q\hat{\mathbf{h}}_Q$ is to be modelled as:

$$\beta_Q\hat{h}_Q=d_1^{-1}+\sum_{j=1}^{Q-1}\beta_j\hat{h}_j \tag{6.4}$$

The model of the target T-vector $\beta_Q\hat{\mathbf{h}}_Q$ consists of unknown parameters $\beta_1,\ldots\ldots,\beta_{Q-1}$. To fully determine the target T-vector, we must determine these unknown parameters according to a certain criterion. Under the present scheme these parameters are determined such that the distance between the target T-vector and the input space **X** is minimized. Mathematically, we would like to minimize the following cost function

$$\|\beta_Q\hat{h}_Q-P_X\beta_Q\hat{h}_Q\|^2 \tag{6.6}$$

where $\mathbf{P_X}$ is the projection operator on the input space **X**. It is explicitly expressed as

$$P_X\beta_Q\hat{h}_Q=\sum_{i=1}^{n}\frac{<\beta_Q\hat{h}_Q\ ,\ x_i^o>}{\|x_i^o\|^2}x_i^o \tag{6.7}$$

where the $\mathbf{x}_i^o$ vectors are the orthogonalized version of the input T-vectors $\mathbf{x}_i$, obtained using the Gram-Schmidt orthogonalization process. Substituting the expression for the target T-vector $\beta_Q\hat{\mathbf{h}}_Q$ into the above expression , we obtain

$$\begin{aligned}
P_X\beta_Q\hat{h}_Q &= \sum_{i=1}^{n} \frac{\langle d_1^{-1}+\sum_{j=1}^{Q-1}\beta_j h_j\ ,\ x_i^o\rangle}{\|x_i^o\|^2} x_i^o \\
&= \sum_{i=1}^{n} \frac{\langle d_1^{-1}, x_i^o\rangle}{\|x_i^o\|^2} x_i^o + \sum_{i=1}^{n}\sum_{j=1}^{Q-1} \frac{\langle \beta_j\hat{h}_j\ ,\ x_i^o\rangle}{\|x_i^o\|^2} x_i^o \\
&= \sum_{i=1}^{n} \frac{\langle d_1^{-1}, x_i^o\rangle}{\|x_i^o\|^2} x_i^o + \sum_{j=1}^{Q-1}\sum_{i=1}^{n} \frac{\langle \beta_j\hat{h}_j\ ,\ x_i^o\rangle}{\|x_i^o\|^2} x_i^o
\end{aligned} \tag{6.8}$$

As can be seen from the above expression, the projection of the target T-vector $\beta_Q\hat{\mathbf{h}}_Q$ onto the input space **X** is equal to the summation of the projections of the inverse desired output T-vector $\mathbf{d}_1^{-1}$ and the projections of all the old hidden T-vectors $\mathbf{h}_j$, j=1 to Q-1, multiplied by their corresponding unknown factor β_j ,onto **X**. Since the various parameters in the above equation including $\mathbf{d}_1^{-1}$, $\mathbf{x}_i^o$ and $\mathbf{h}_j$ are known, the above equation is a linear equation of the unknown $\beta_1, \ldots, \beta_{Q-1}$. Finally, we would evaluate the expression $\beta_Q\hat{\mathbf{h}}_Q - \mathbf{P}_X\beta_Q\hat{\mathbf{h}}_Q$:

$$\begin{aligned}
&\beta_Q\hat{h}_Q - P_X\beta_Q\hat{h}_Q \\
&= \beta_Q\hat{h}_Q - \Big(\sum_{i=1}^{n} \frac{\langle d_1^{-1}, x_i^o\rangle}{\|x_i^o\|^2} x_i^o + \sum_{j=1}^{Q-1}\sum_{i=1}^{n} \frac{\langle \beta_j h_j\ ,\ x_i^o\rangle}{\|x_i^o\|^2} x_i^o\Big) \\
&= d_1^{-1} + \sum_{j=1}^{Q-1}\beta_j h_j - \Big(P_X d_1^{-1} + \sum_{j=1}^{Q-1} P_X\beta_j h_j\Big) \\
&= d_1^{-1} - P_X d_1^{-1} + \sum_{j=1}^{Q-1}\beta_j h_j - \sum_{j=1}^{Q-1}\beta_j P_X h_j \\
&= d_1^{-1} - P_X d_1^{-1} + \sum_{j=1}^{Q-1}\beta_j (h_j - P_X h_j)
\end{aligned} \tag{6.9}$$

Our objective is that we should minimize the squared norm of the above expression,

$$\begin{gathered}\min \|\beta_Q \hat{h}_Q - P_X \beta_Q \hat{h}_Q\|^2 = \|e_{xd} + \sum_{j=1}^{Q-1} \beta_j e_{xh}^j\|^2 \\ \textit{where} \quad e_{xd} \triangleq d_1^{-1} - P_X d_1^{-1} \\ e_{xh}^j \triangleq h_j - P_X h_j\end{gathered} \tag{6.10}$$

The above cost function can be minimized if we choose $-\Sigma_j\, ß_j \mathbf{e}_{xh}^{j}$ to be equal to the projection of $\mathbf{e}_{xd}$ onto the space spanned by the T-vectors $\mathbf{e}_{xh}^{j}$, which we denoted as **E**. In other words,$-\Sigma_j\, ß_j \mathbf{e}_{xh}^{j}$ should be equal to $\mathbf{P}_E\mathbf{e}_{xd}$ in order for the distance between the target T-vector $ß_Q\hat{\mathbf{h}}_Q$ and the input space **X** to be minimized. Since the projection operator on **E** is given by

$$P_E e_{xd} = \sum_{j=1}^{Q-1} \frac{<e_{xd}, e_{xh}^{oj}>}{\|e_{xh}^{oj}\|^2} e_{xh}^{oj} = \sum_{j=1}^{Q-1} \alpha_j e_{xh}^{oj} \tag{6.11}$$

where the T-vectors $\mathbf{e}_{xh}^{oj}$ is the orthogonalized version of $\mathbf{e}_{xh}^{j}$ obtained using the Gram-Schmidt process. As a result, the coefficients α_j above can be obtained as the normalized scalar product between $\mathbf{e}_{xd}$ and $\mathbf{e}_{xh}^{oj}$. The original coefficients $ß_j$ can be obtained as a linear combination of α_j by the following inversion algorithm on the Gram-Schmidt process.

The Inversion Algorithm for the Gram-Schmidt Process: Given a set of vectors $\mathbf{v}_i$, i=1 to n, and their orthogonalized counterpart $\mathbf{u}_i$, i=1 to n, and given an arbitrary linear combination of the orthogonalized vectors $\mathbf{w} = \Sigma_i\, \alpha_i \mathbf{u}_i$, the stated algorithm finds the coefficients $ß_i$, i=1 to n which expresses **w** as a linear combination of the unorthogonalized vectors $\mathbf{v}_i$, i.e., $\mathbf{w} = \Sigma_i\, ß_i \mathbf{v}_i$.

As mentioned in Chapter 5, the orthogonalized vectors $\mathbf{u}_i$ are related to the unorthogonalized vectors $\mathbf{v}_i$ by the following relation:

$$\begin{aligned} &u_1 = v_1 \\ &u_i = v_i - P_{i-1} v_i \qquad 2 \le i \le n \end{aligned} \tag{6.12}$$

From the above expressions and from the definition of $\mathbf{P}_{i-1}$, it is seen that $\mathbf{u}_k$ can be expressed as the linear combination $\mu_{k1}\mathbf{v}_1 + \ldots \mu_{kk}\mathbf{v}_k$, for $k \le n$ since $\mathbf{P}_{k-1}\mathbf{v}_k$ is a function of the T-vectors $\mathbf{v}_1 \ldots \mathbf{v}_{k-1}$. In other words , the unorthogonalized vector $\mathbf{v}_k$ is contained in the linear combination expression of $\mathbf{u}_l$, $k \le l \le n$. In view of the relationship between the orthogonalized vectors and the unorthogonalized vectors, the coefficients μ_{ki}, can be recursively determined as below:

$$\begin{aligned} &For\ 1 \le i \le n \\ &\mu_{ii} = 1 \\ &\mu_{ki} = -\sum_{m=i}^{k-1} \frac{<v_k, u_m>}{\|u_m\|^2} \mu_{mi} \qquad i \le k \le n \end{aligned} \tag{6.13}$$

Therefore, the linear combination coefficients β_i for the unorthogonalized vectors $\mathbf{v}_i$ can be obtained as a function of the coefficients obtained above:

$$\beta_i = \sum_{m=1}^{n} \alpha_m \mu_{mi} \qquad 1 \le i \le n \tag{6.14}$$

From the inversion algorithm, one can obtain the coefficients β_i in the linear combination expression for $\mathbf{P}_E\mathbf{e}_{xd}$ in terms of its unorthogonalized T-vectors $\mathbf{e}_{xh}^{\ j}$. In other words, we obtain the estimation for the hidden T-vector for the new hidden node since the same coefficients β_j can be substituted into the expression for the model for the optimum target T-vector $\beta_Q \hat{\mathbf{h}}_Q^{\ *}$.

In summary, when we would like to evaluate the coefficients β_j in Eq(6.10), we are dealing with a linear least square problem in which these parameters are determined to minimize the squared norm in the expression. Our method of obtaining these coefficient, by first orthogonalizing the $\mathbf{e}_{xh}^{\ j}$'s, obtain the scalar product α_j, and then calculate the original coefficient β_j in the least square expression, is equivalent to the **QR**-process of finding a least square solution in linear algebra [41]. We have phrased the **QR**-process in the above form such as to give a physical meaning to the intermediate results of the **QR**-process as the orthogonalized version of the basis spanning the space on which the vector to be approximated is projected such that the projection operator can be expressed in a particular simple form which involves only the normalized scalar product between the vector to be approximated and the various basis vectors.

Hereafter, whenever the projection $\mathbf{P_H v}$ is represented as a linear combination of the unorthogonalized T-vectors $\mathbf{v}_i$, we may denote the linear combination coefficient for the j-th basis as $(\mathbf{P_H v})_j$, with full understanding that we first obtain the coefficients in terms of the orthogonalized T-vectors and then apply the inversion algorithm to obtain the coefficients in terms of the unorthogonalized T-vectors.

The remaining steps in the hidden node initialization scheme involves the extraction of the actual hidden T-vector $\mathbf{h}_Q$ from the optimum target T-vector $\beta_Q \hat{\mathbf{h}}_Q^*$, the calculation of the hidden weights W_{iQ} for the new hidden node, and calculation of the output weights u_{jl} $1 \leq j \leq Q$ for all the hidden nodes. However, additional adjustment steps are required for the new hidden T-vector in addition to the above mentioned straightforward calculations in order to provide a favourable initial state for the BP fine tuning process.

6.5.3 Preparation for the BP fine tuning process

The present algorithm attempts to assign the new hidden and output weights deterministically and in such a way that the appended network is relatively close to the global minimum of the current restricted network due to the error minimization strategy introduced in the subsection 6.5.2.3. However, a BP fine-tuning process, i.e. applying the classical BP algorithm to the appended network immediately after the new hidden node initialization, is indispensable for the completeness of the algorithm due to the various approximation assumption made during the derivation of the node addition strategy. These include:

(1) The linear approximation for the sigmoid function. The BP fine-tuning process will appropriately introduce controlled degrees of non-linearity to the new hidden node

whenever the linear representation is not adequate.

(2) The approximate assignment of β_Q from the optimum target T-vector $\beta_Q\hat{\mathbf{h}}_Q^*$: it will be seen later that our criterion of assignment for β_Q consists of only ensuring that each component of the resulting optimum target hidden T-vector $\hat{\mathbf{h}}_Q^*$ does not exceed a certain limit such that the linearity assumption for the sigmoid function is not violated. As a result, the assignment of the hidden T-vector $\hat{\mathbf{h}}_Q$ is only an approximate process and has to be further fine-tuned by an additional optimization procedure.

Therefore, the necessity for the BP fine-tuning process is apparent for the completeness of the deterministic algorithm. We can carry out the fine-tuning process by calculating all the relevant parameters for the new hidden node and apply the classical BP process to the network. However, these deterministic assignments do not necessarily constitute a favourable initial state for the BP fine-tuning process. We should instead pre-process the parameters of the new node such that the fine-tuning process can be carried out with the greatest efficiency. The preparation step for the BP process concerns two aspects:

(1) The suitability of target T-vector $\beta_Q\hat{\mathbf{h}}_Q$ as an initial state for the fine-tuning process: It is seen that the evaluation process leading to the optimum target T-vector $\beta_Q\hat{\mathbf{h}}_Q^*$ for the new hidden node is wholly deterministic, and there is no guarantee that the BP fine-tuning process will operate effectively at this state : the criterion of effectiveness being that the gradient of the error surface at most training patterns should not be excessively small such that the BP process can make a real contribution towards reducing the error of the network. Otherwise, when the gradient of the error surface are small, the BP process cannot make a true descent on the error surface, despite the addition of a new hidden unit, and the error of the network will remain more or less the same, i.e., the node addition has not served its purpose of further reducing the error of the network. As a result, the first preprocessing step for the new hidden node should satisfy the following two criterion:

(a) It should adjust the optimum target T-vector $\beta_Q\hat{\mathbf{h}}_Q^*$ such that the resulting optimum target T-vector $\beta_Q\hat{\mathbf{h}}_Q^*$ should induce a reasonably large gradient at the error surface at most training patterns.

(b) The pre-processing step should not disturb the state of optimality induced by the previous deterministic assignment.

The idea of the pre-processing step originates from the observation that there is one condition in which the error gradient induced by the new hidden node will be small at most training patterns and that is the case when the disparity between the number of positive components and the number of negative components in the optimum target T-vector $\beta_Q\hat{\mathbf{h}}_Q^*$ is large. Assuming for the moment that the sigmoid function of the new hidden node has a zero average value and denote this function by f_o . i.e. $f_o(x)=f(x)-0.5$. We have introduced this assumption because in the previous process of determining the target T-vector which is closest to the input space **X**, we should have formally subtracted 0.5 from the target vector since , although we have assumed the near linearity of the sigmoid function, the resulting function still has an offset of 0.5 which is to be catered for. However, at this stage we temporarily assume the replacement of f by f_o such that the output of the new hidden node can be described more conveniently in terms of positive or negative entities instead of values which are greater or less than 0.5. Assuming that the optimum target T-vector $\beta_Q\hat{\mathbf{h}}_Q^*$ is approximated reasonably accurately by the new hidden node, the resulting sigmoid function when superimposed on the training set domain will give the following picture.

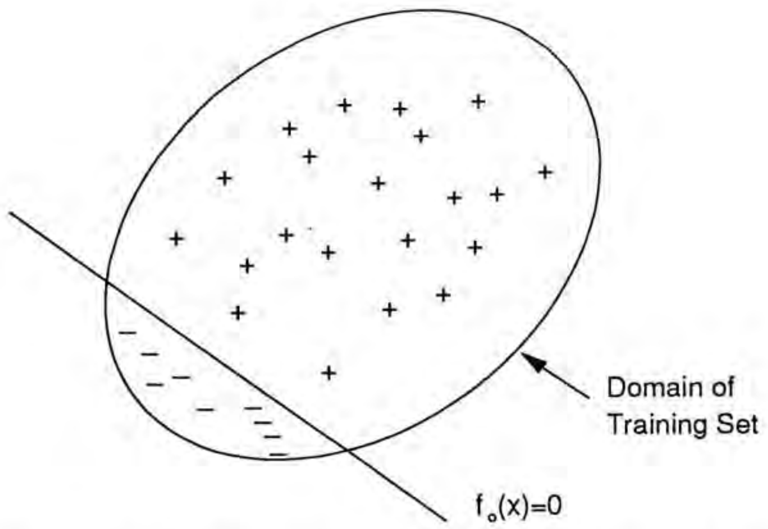

Fig 6.4 Relationship of the new sigmoid surface to the training set domain before pre-processing

It is seen that due to the more numerous positive components contained in target T-vector $ß_Q\hat{\mathbf{h}}_Q^*$, the locus $f_o(x) = 0$ of the sigmoid hypersurface has shifted almost to the periphery of the training set domain. As the gradient of the sigmoid surface is greatest in the vicinity of the locus $f_o(x) = 0$, and this region has been shifted away from the centre of the training set domain, most of the training patterns will be under the saturation region of the sigmoid hypersurface which possesses a small gradient. As the hidden weight adaptation term in the BP equation is directly proportional to the sigmoid surface gradient at the various training patterns, the adaptation of the hidden weight for the new node will be small for most of the training patterns under the above situation.

Fortunately, a single shifting of all the components of the target T-vector $ß_Q\hat{\mathbf{h}}_Q^*$ will remedy the above problem. Let h^+ be the average value of the positive components of $ß_Q\hat{\mathbf{h}}_Q^*$ and h^- the negative average value for the components. Then we can apply the following shifting procedure to each component of $ß_Q\hat{\mathbf{h}}_Q^*$, for $1 \le t \le p$

$$\begin{aligned} \beta_Q\hat{h}^*_{Qs}(t) &= \beta_Q\hat{h}^*_Q(t) + \frac{h^+ + h^-}{2} \qquad & |h^+| < |h^-| \\ &= \beta_Q\hat{h}^*_Q(t) - \frac{h^+ + h^-}{2} \qquad & |h^+| > |h^-| \end{aligned} \tag{6.15}$$

The purpose of the above shifting procedure tends to "equalize" the number of positive components and negative components of $ß_Q\hat{\mathbf{h}}_Q^*$. The relationship of the new sigmoid surface to the training set domain after pre-processing is shown in the fig 6.5:

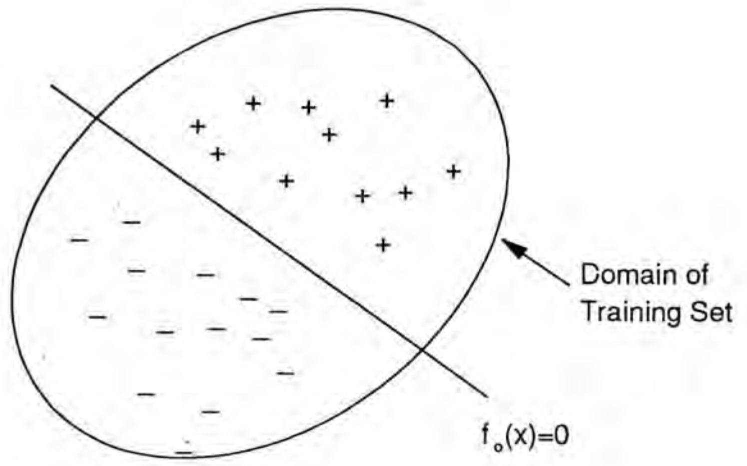

Fig 6.5 Relationship of the new sigmoid surface to the training set domain after pre-processing

From the above diagram it is seen that most of the training patterns are under the "steep" region of the sigmoid surface. As a result, the BP process will induce a non-negligible adaptation for the new hidden weights for most of the training patterns which effects the searching of the global minimum in the restricted network.

This shifting of $\beta_Q \hat{\mathbf{h}}_Q^*$ will not cause any disturbance to the optimality ensured by the previous deterministic assignment process since the bias vector $\mathbf{x}_1$ is a member of the basis spanning the input space **X**. As the shifting of $\beta_Q \hat{\mathbf{h}}_Q^*$ involves the addition of a bias vector to $\beta_Q \hat{\mathbf{h}}_Q^*$ and we would like to maintain the previous relationship between $\beta_Q \hat{\mathbf{h}}_Q^*$ and **X**, we would simply add the same bias vector to the projection of $\beta_Q \hat{\mathbf{h}}_Q^*$ on **X** in order to maintain the above relationship. Therefore the shifting process satisfies criteria (a) and (b) mentioned above and qualifies as a valid pre-processing step.

6.5.4 Determination of the Target Hidden T-vector

With the determination of the shifted target T-vector $\beta_Q \hat{\mathbf{h}}_{Q\ s}^*$, we obtain an estimation for $\hat{\mathbf{h}}_Q^*$ according to the relation below

$$\hat{h}_Q^* = \frac{\beta_Q \hat{h}_{Qs}^*}{\hat{h}_{Qs,\max}^*} \times 0.3 + 0.5$$
$$\hat{h}_{Qs,\max}^* = \max |\hat{h}_{Qs}^*(t)| \qquad 1 \le t \le p \tag{6.16}$$

The s subscript indicates the shifted T-vectors with respect to the original T-vectors prior to the pre-processing step. In Eq (6.16), we have included the factor 0.3 as we have to ensure that the optimum target hidden T-vector $\hat{\mathbf{h}}_Q^*$ is within the range [0.2,0.8]. As we have ignored this offset in the determination of the target T-vector, we have to compensate by adding 0.5 to our previous target T-vector. In order to satisfy the above magnitude criterion, we have to restrict the magnitude of the target T-vector to [-0.3 ,0.3], which is the reason for the factor 0.3 in Eq (6.16).

By utilizing the above relationships, we can ensure that the norm of the T-vector $\hat{\mathbf{h}}_Q^*$ stays well within the norm in which the linearity assumption for the hidden node remains valid.

6.5.5 Determination of the Hidden Weights

After obtaining the optimum target hidden T-vector $\hat{\mathbf{h}}_Q^*$, we can evaluate the hidden weights w_{iQ} of the new node in a straightforward way. Define

$$\hat{h}_Q^{*-1} = f^{-1}(\hat{h}_Q^*) \tag{6.17}$$

The hidden weights of the new node is given by

$$w_{iQ} = (\boldsymbol{P}_X \hat{h}_Q^{*-1})_i \tag{6.18}$$

From the hidden weights of the new node, we can thus determine the hidden T-vector $\mathbf{h}_Q$ of the new node by propagating the input T-vectors through the new hidden node.

6.5.6 Determination of the Output Weight

With the availability of the hidden T-vectors $\mathbf{h}_1$,....., $\mathbf{h}_Q$ and the inverse desired output T-vector $\mathbf{d}_1^{-1}$, the output weights of the network u_{Q1} of the whole network can simply be determined as

$$u_{jl}=(\boldsymbol{P}_H \boldsymbol{d}_1^{-1})_j \tag{6.19}$$

Similarly, for a multi-output network with inverse desired output T-vectors $\mathbf{d}_1^{-1}$,....,$\mathbf{d}_k^{-1}$, the output weights u_{jk} are determined from Eq (6.19) with the subscript 1 replaced by k

$$u_{jk}=(\boldsymbol{P}_H \boldsymbol{d}_k^{-1})_j \tag{6.20}$$

6.6 Linear Independency Assurance for the New Hidden T-Vector

The procedure for ensuring the linear independency of the new hidden T-vector would now be given. This procedure is instrumental in ensuring the convergence of the overall algorithm as mentioned in the first section of this Chapter. First we would like to introduce some terminologies: we define the set of T-vectors $\mathbf{n}_1$, ..., $\mathbf{n}_p$,which we call the neural basis,as

$$\begin{aligned} \boldsymbol{n}_j&=[n_{j1},...,n_{jt},...,n_{jp}]^T \qquad 1\leq j\leq p \\ n_{jt}&=1 \qquad j=t \\ &=0 \qquad j<t \end{aligned} \tag{6.21}$$

and the set of vectors $\mathbf{e}_1$, ... $\mathbf{e}_p$, which we call the standard basis as

$$\begin{aligned} e_j&=[e_{j1},\dots,e_{jt},\dots,e_{jp}]^T && 1\leq j\leq p \\ e_{jt}&=1 && j=t \\ &=0 && j\neq t \end{aligned} \tag{6.22}$$

It is seen that both sets of vectors span the Euclidean space $\mathbf{E}^p$. The set of vectors $\mathbf{n}_j$ is particular relevant to our present discussion as each $\mathbf{n}_j$ can be reproduced by a hidden node by applying Eq (3.1) with a large scale-up factor to the j-th pattern after the pattern is arranged in Euclidean distance. In general, if the training pattern is not ordered according to the Euclidean distance and the new hidden node is initialized with Eq (3.1), the hidden T-vector of the new node will be a permuted version of $\mathbf{n}_j$. Nevertheless, we will here after refer to these permuted version under the same name $\mathbf{n}_j$, as the present algorithm does not require the ordering of the training set according to the Euclidean distance. The growth algorithm mentioned in Chapter 4, on the other hand, produces hidden node which emit the $\mathbf{e}_j$'s such that isolation of the "difficult" training pattern is possible.

Suppose that at a certain stage of node addition the resulting new hidden T-vector $\mathbf{h}_Q$ is linearly dependent on the old hidden T-vectors and Q is smaller than p, then we can replace $\mathbf{h}_Q$ by an appropriate member of $\mathbf{n}_j$ which is linearly independent from the old hidden T-vectors, since the old hidden space $\mathbf{H}$ spanned by $\mathbf{h}_1, \dots, \mathbf{h}_{Q-1}$ contains at most (Q-1) of the $\mathbf{n}_j$ which is smaller than p. The question is : what $\mathbf{n}_j$ should we choose? Though we can choose any $\mathbf{n}_j$ which is linearly independent from the old hidden T-vector set, we are clearly sacrificing the optimality condition imposed by the previous node initialization scheme by abruptly pursuing for the linear independency condition. Instead we should adopt the procedure below:

Whenever the linear dependency condition is encountered, remove 10% of the components of each T-vector involved in the node initialization procedure and repeat the procedure. The components removed are those invoking the smallest error at the network output such that the resulting node initialization procedure with the reduced T-vectors still cater sufficiently for those patterns invoking large errors. This progressive reduction procedure is suggested because the present node initialization scheme depends on multiple training patterns for its optimality, while the satisfaction of linear independency condition depends on only one training pattern (which corresponds to the particular $\mathbf{n}_j$ chosen). Therefore, it would be reasonable to reduce the training set progressively such that the transition from optimality to linear independency would be gradual. The reduction should be implemented in such a way that the training set is reduced pattern by pattern until 10% of the pattern are removed while the linearly independent condition is checked at each stage. If the reduced old hidden T-vectors ever become linearly dependent, replace the components removed previously and remove that pattern with the next smallest error instead.

Suppose that the dimension of all the T-vectors involved in the optimization process is reduced to Q, the number of hidden nodes in the appended network, and the new hidden vector $\mathbf{h}_Q$ is still linearly dependent on the old hidden T-vectors. At this stage the old hidden T-vectors $\mathbf{h}_1,......,\mathbf{h}_{Q-1}$ must be linearly independent from at least one of the neural basis $\mathbf{n}_j$,since the Q-1 old hidden T-vectors cannot span the whole space $\mathbf{E}^Q$. We could then pick all of those neural basis which are linearly independent from the old set of hidden T-vectors (this could be checked using the Gram-Schmidt process) and choose among these linearly independent basis the one which is closest to the optimum target T-vector $\beta_Q\hat{\mathbf{h}}_Q^*$. We could now simply choose this neural basis as our new hidden T-vector and initialize the new hidden node accordingly to Eq (3.1) with a large enough scale up factor. The above last measure serves mainly as a precautionary measure as it is observed in simulation studies that linear independency is achieved long before the dimension of each T-vector is reduced beyond Q.

Finally, if the new hidden T-vector is found to be linearly dependent on the old hidden T-vectors, we should first record the present state of the network and apply the BP fine-tuning process, since there may be a chance that the resulting hidden T-vectors after the fine-tuning process may become linear independent. It was only when this process fails that we restore the previous state of the network and apply the progressive reduction process.

6.7 Extension to the Multi-output Case

In the previous chapter it has been briefly mentioned how we should adapt the present algorithm to the multi-output neural network: we should transform the inverse output

T-vector $\mathbf{d}_k^{-1}$ into another set of T-vectors $\mathbf{c}_k^{-1}$ which spans the same space $\mathbf{D}^{-1}$ and which are much closer to the input space **X**.

The above purpose can be achieved if we first project each $\mathbf{d}_k^{-1}$ onto the input space **X** and then re-project the resulting projection $\mathbf{P}_X\mathbf{d}_k^{-1}$ back onto the space $\mathbf{D}^{-1}$, i.e., we would like to evaluate $P_{D^{-1}}P_X d_k^{-1}$ as depicted in the following diagram

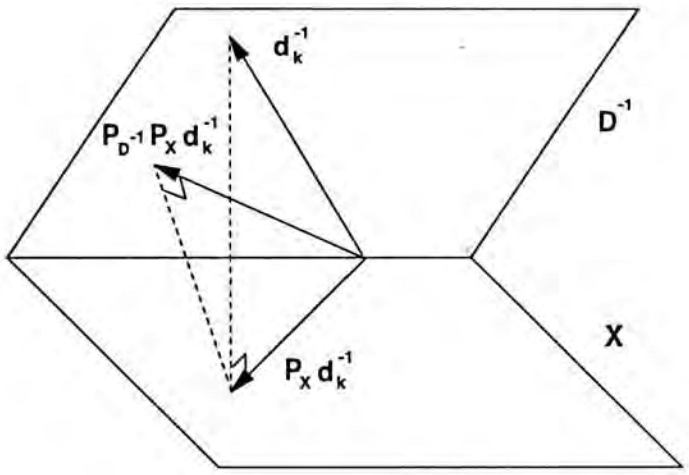

Fig 6.6 The evaluation of the double projection vector

It can be seen in the above diagram that the resulting double projected T-vector is in general much closer to the input space **X** than the original T-vector $\mathbf{d}_k^{-1}$. The double projection vector can be made closer to the input space **X** by adopting the following approach, we first form the T-vector $\mathbf{f}_k$ which is the difference between $\mathbf{d}_k^{-1}$ and the double projection vector.

$$f_k = d_k^{-1} - P_{D^{-1}} P_X d_k^{-1} \tag{6.23}$$

and construct the following T-vector

$$c_k^{-1} = d_k^{-1} + \sigma f_k \tag{6.24}$$

We can subsequently search for the parameter σ which minimizes the distance between $\mathbf{c}_k^{-1}$ and its projection onto the input space $\mathbf{X}$, $\mathbf{P}_X\mathbf{c}_k^{-1}$, i.e., we would like to minimize $\| \mathbf{c}_k^{-1} - \mathbf{P}_X\mathbf{c}_k^{-1} \|^2$. The above construction is depicted in the following diagram.

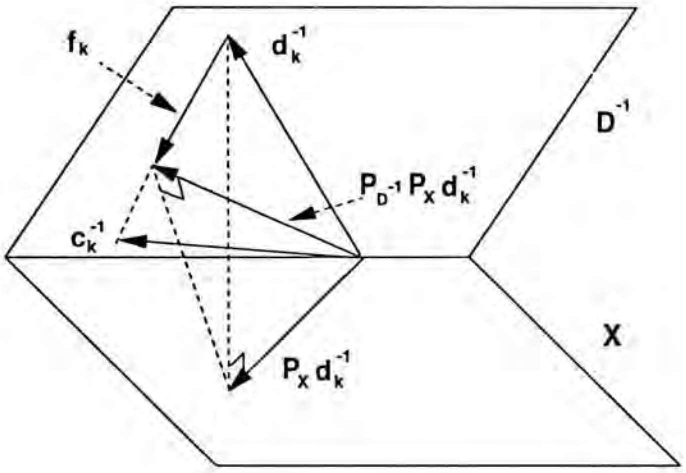

Fig 6.7 The construction of $\mathbf{c}_k^{-1}$

It can be seen in the above diagram that the resulting $\mathbf{c}_k^{-1*}$ is closer to the input space $\mathbf{X}$ than the original double projection vector. The form of the cost function as described above which involves the minimization of the distance between a T-vector and its projection on the input space $\mathbf{X}$ is identical to that involved in the optimization of target T-vector $\beta_Q\hat{\mathbf{h}}_Q^*$. Therefore, identical methods can be used to optimize the current cost function for each $\mathbf{d}_k^{-1}$ such that we obtain a set of new T-vectors $\mathbf{c}_k^{-1}$ which is much closer to $\mathbf{X}$.

An additional issue to be considered is the linear independency of the new set of T-vectors $\mathbf{c}_k^{-1}$. To ensure the linear independency of the new T-vectors, we can include a backtracking process for each $\mathbf{c}_k^{-1}$. After the evaluation of each $\mathbf{c}_k^{-1}$, we check for the linear independency of the whole set of T-vectors, $\mathbf{c}_1^{-1}, \ldots, \mathbf{c}_k^{-1}, \mathbf{d}_{k+1}^{-1}, \ldots, \mathbf{d}_m^{-1}$ using the Gram-Schmidt orthogonalization procedure. If the linear independency criterion is satisfied, we retain the current $\mathbf{c}_k^{-1}$ as member of the new set of vectors. Otherwise, we backtrack through the parameter σ:

$$c_k^{-1}=d_k^{-1}+i\sigma f_k \qquad i=0.9,0.8,.....,0.1,0 \tag{6.25}$$

It is seen that with i=0, the $\mathbf{c}_k^{-1}$ reverts to the former vector $\mathbf{d}_k^{-1}$, which by definition is linear independent from all vectors in the partially converted set, and therefore in the course of backtracking there exists an i such that the resulting $\mathbf{c}_k^{-1}$ is linearly independent from the rest of the set.

Finally we represent each $\mathbf{c}_k^{-1}$ with a cluster of hidden nodes according to the single-output network construction scheme. The cluster of hidden nodes which represents a particular $\mathbf{c}_k^{-1}$ is labelled with double subscript $\hat{\mathbf{h}}_{kj}$ such that the first subscript identifies the cluster of hidden nodes approximating the $\mathbf{c}_k^{-1}$ and the second subscript identifies each hidden node in the cluster. By virtue of the above conversion of basis, each cluster would contain less nodes than it would have been if it directly approximates each $\mathbf{d}_k^{-1}$. Finally, instead of approximating each $\mathbf{c}_k^{-1}$ in isolation, we can amalgamated these approximation procedures into a single node addition procedure according to the node addition sequence described below

$$\hat{h}_{11},\hat{h}_{21},...,\hat{h}_{m1},\hat{h}_{12},\hat{h}_{22},.....$$

and apply the BP fine-tuning procedure after the addition of each node. In this way we allow the hidden space $\mathbf{H}$ to expand quickly in all its major linear independent directions. During the approximation of each $\mathbf{c}_k^{-1}$, we do not have to restrict ourselves to consider only $\mathbf{h}_{k1}$, $\mathbf{h}_{k2}$, ..., as our old hidden T-vector set. The inclusion of all other hidden nodes in the optimization process of $\beta_{kQ}\hat{\mathbf{h}}_{kQ}$ provides more degrees of freedom to the approximation process. In other words, each stage of node addition resembles the node addition process of the single-output network except with different $\mathbf{c}_k^{-1}$ at each stage.

6.8 Convergence Proof for the Deterministic Algorithm

In view of the course of development for the present deterministic algorithm, the convergence proof for the algorithm is almost self-evident. Assuming that for a multi-output neural network with inverse desired output T-vector $\mathbf{d}_1^{-1}$, ..., $\mathbf{d}_m^{-1}$, and assuming that the p-th hidden node is added to the network and the corresponding new hidden T-vector, by virtue of the linearly independent assurance scheme, is linearly independent from the old hidden T-vectors. As a result the whole set of hidden t-vectors form a basis for the space $\mathbf{E}^p$. As a result, the neural network representation criterion mentioned in Chapter 5 is satisfied and the corresponding training set can be exactly represented by the neural network.

6.9 The Flow of the Deterministic Dynamic Node Creation Algorithm

We may now capitulate on the flow of the deterministic dynamic node creation algorithm by the following flowchart:

6.10 Experimental Results and Performance Analysis

(1) Parity Problem

The deterministic algorithm is first applied to the parity problem as in the case for the growth algorithm. The node addition monitoring scheme for the present algorithm is identical to that for the growth algorithm and we would not repeat it here. The training parameters and convergence criterion used are the same as in the previous simulation studies.

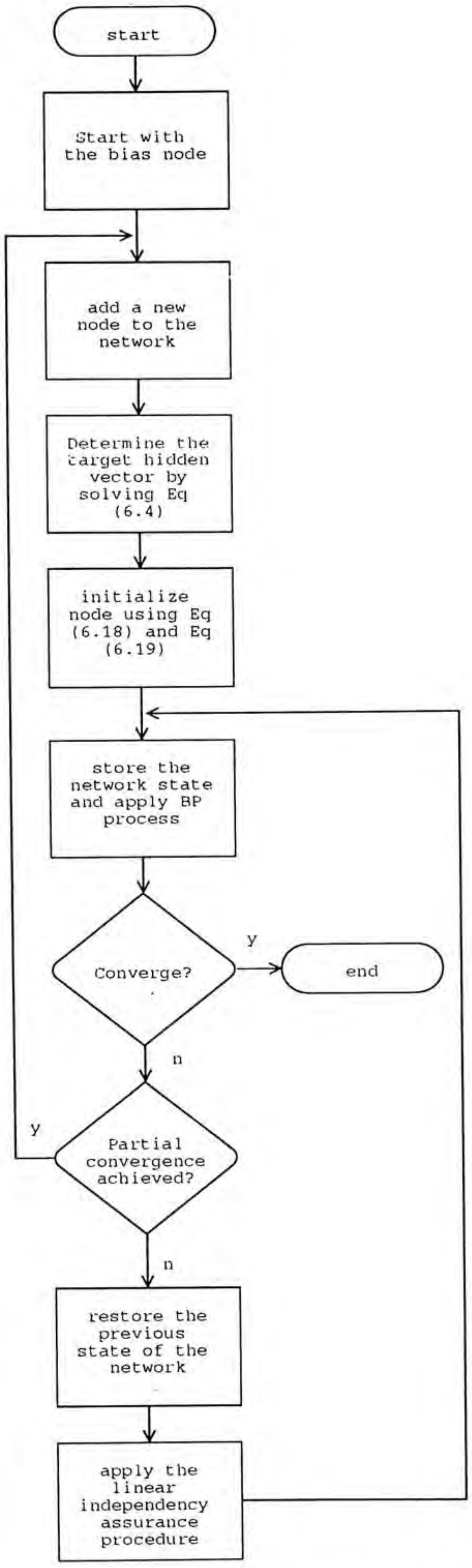
start
Start with the bias node
add a new node to the network
Determine the target hidden vector by solving Eq (6.4)
initialize node using Eq (6.18) and Eq (6.19)
store the network state and apply BP process
Converge?
y
end
n
Partial convergence achieved?
y
n
restore the previous state of the network
apply the linear independency assurance procedure

Fig 6.8 Flow of the deterministic dynamic node creation algorithm

Parity Order	Deterministic Algorithm		BP Algorithm	
	No. of Epochs	No. of Nodes	No. of Epochs	No. of Nodes
2	21	2	107(0%)	2
3	228	3	62(0%)	3
4	296	5	126(20%)	5
5	457	7	167(20%)	7
6	585	6	583(60%)	6
7	405	7	181(60%)	7

Table 6.2 The deterministic algorithm applied to the parity problem

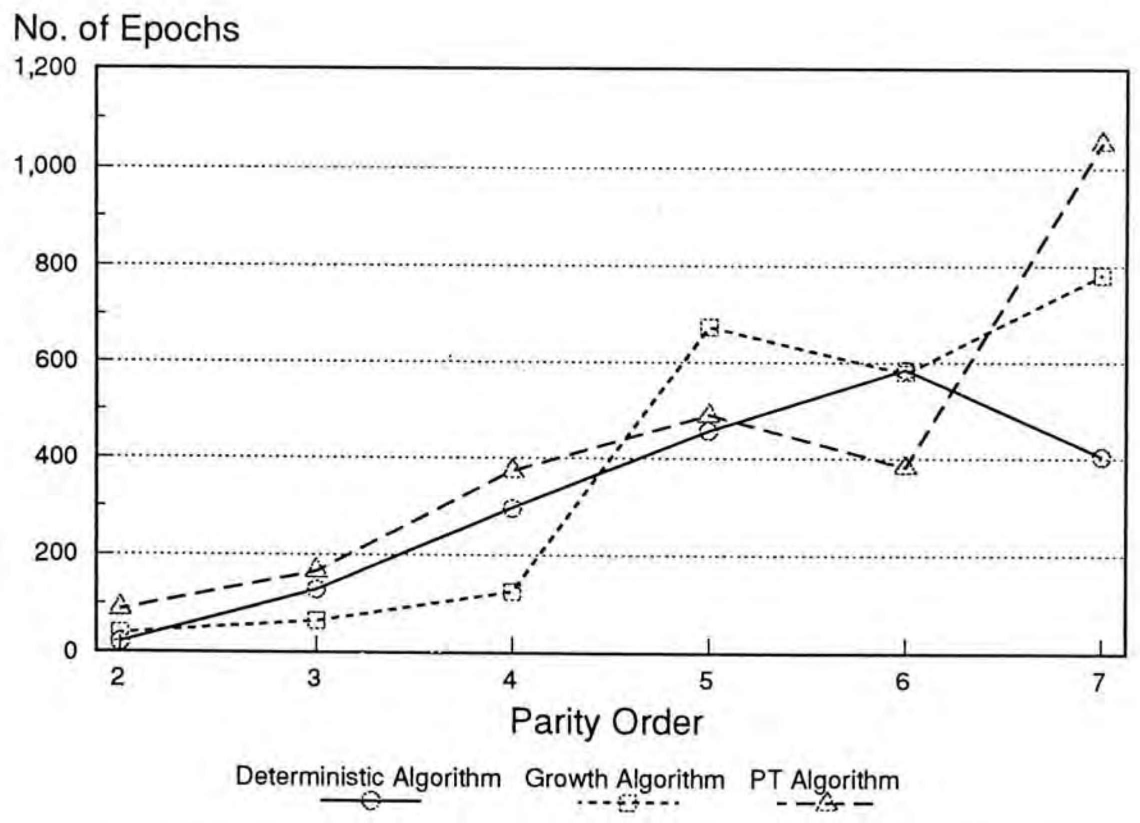

Fig 6.9 Speed comparison between the three dynamic node creation algorithms (parity problem)

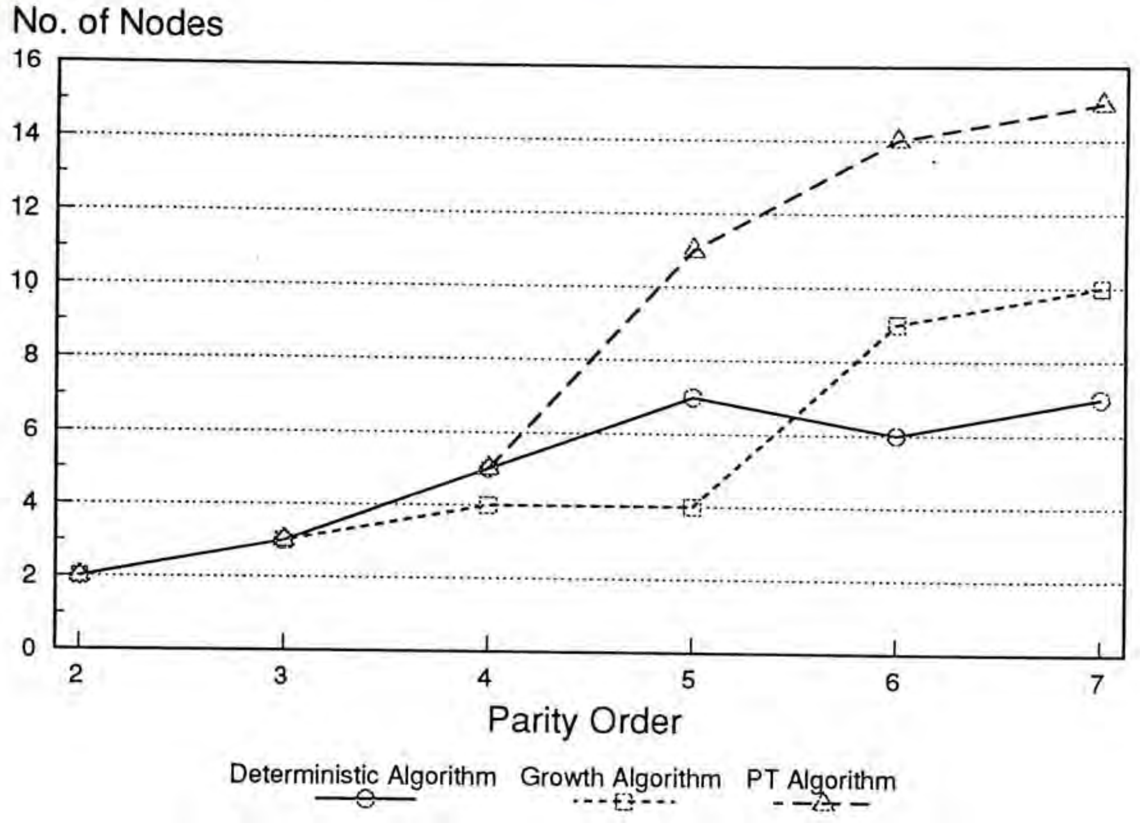

Fig 6.10 Network size comparison between the three dynamic node creation algorithms (Parity problem)

The figure in the parentheses in Table 6.2 indicates the failure rate among 10 trials of BP. It is noticed from the above results that the present algorithm does not encounter any local minima. Moreover, the hidden layer size estimated by present algorithm is smaller than that for the previous 2 algorithms, and the network size approaches the optimal network size even for high-order parity problems, thus illustrating that the present algorithm is an even more accurate hidden layer size estimator than the previous 2 algorithms. In addition, there are sharp rises in the number of training epochs required for both the growth and PT glaorithm with the complexity of the training set, while the rise in the case of the deterministic algorithm remains steady, indicating that its global node initialization scheme reflects more accurately the network's internal status than for local node initialization scheme. From these results, we can once more appreciate the important function of dynamic node creation: in this example, since the present algorithm gives a lower estimate on the hidden layer size, we have used the corresponding number of nodes in the BP networks. The failure rate immediately rises dramatically when compared with the BP networks of Chapter 4 in which larger networks are used. We can thus imagine the difficulties of training a network if we have underestimated the hidden layer size of a conventional BP network.

2. The Handwritten Character Recognition Problem

The present deterministic algorithm is then applied to the same character recognition problem as in Chapter 4. The same training parameters and convergence criterion is applied to the present training situations. This training set also serves as a good testing example for the multi-output version of the deterministic algorithm as the required network network contains 36 output nodes. The results of the simulation studies are presented below:

Error Gradient Threshold	Deterministic Algorithm		BP Algorithm	
	No. of Epochs	No. of Nodes	No. of Epochs	No. of Nodes
0.05%	1189	30	42	30
0.06%	1365	26	53	26
0.07%	960	27	46	27
0.08%	850	26	53	26
0.09%	818	27	46	27
0.10%	656	27	46	27

Table 6.3 The deterministic algorithm applied to the handwritten character recognition problem (training speed and hidden layer size comparison)

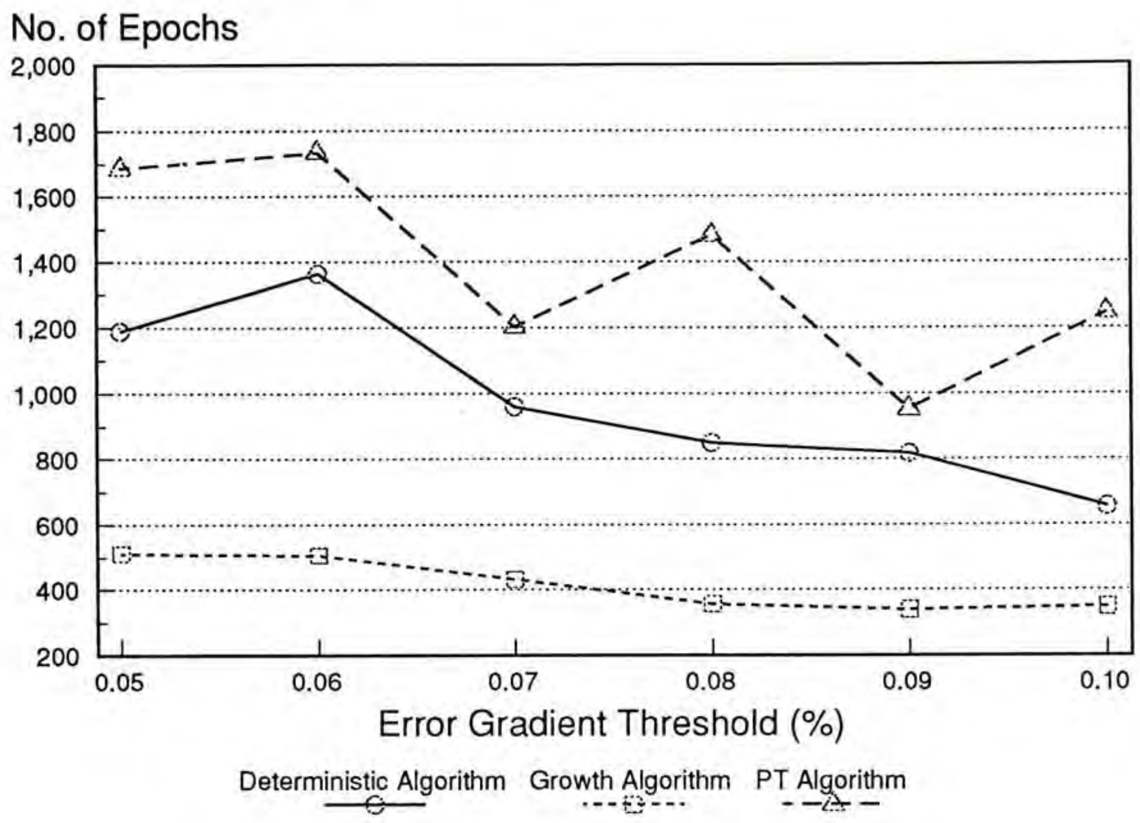

Fig 6.11 Speed comparison between the three dynamic node creation algorithms (Handwritten character recognition problem)

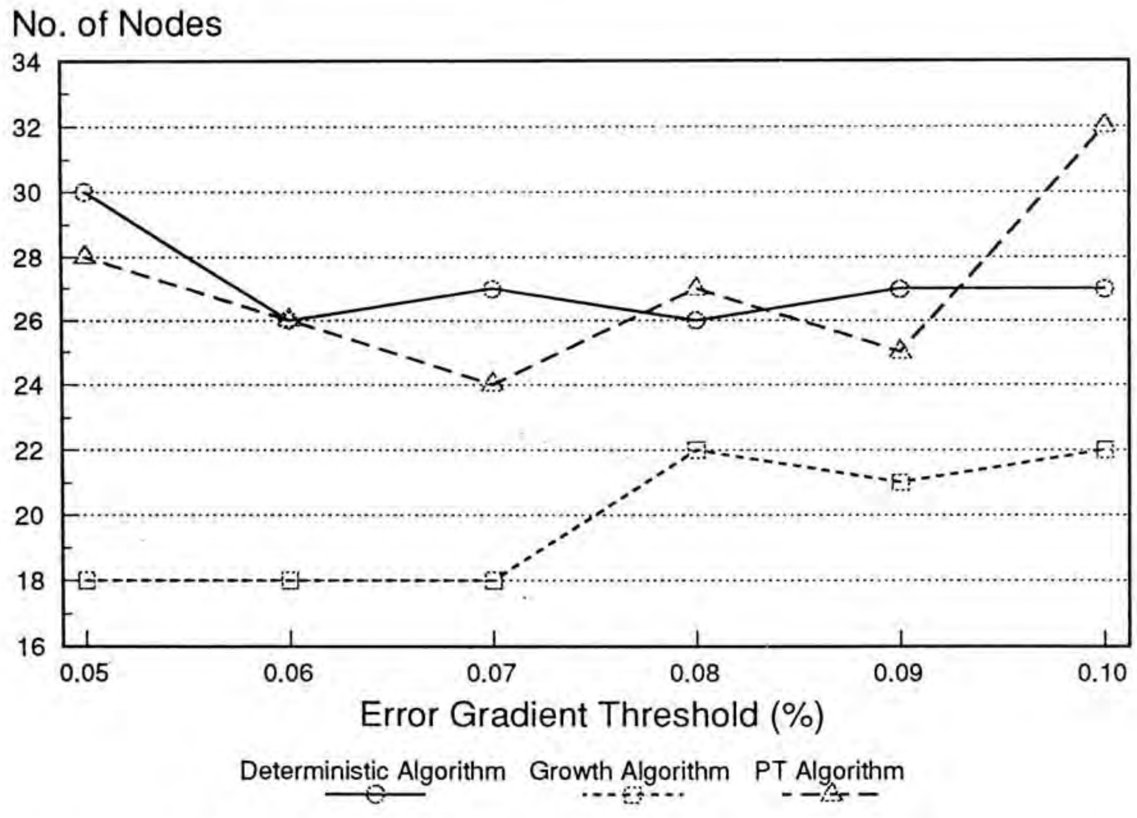

Fig 6.12 Network size comparison between the three dynamic node creation algorithms (Handwritten character recognition problem)

From the above results it is seen that contrary to its behaviour in the parity problem, the resulting network for this algorithm is greater than the one estimated by the growth algorithm while almost the same size as the one created by the PT algorithm. This may be due to the reason that the convergence criterion for this problem involved a fixed sum-squared error threshold of 30. Experimentally, it is observed that the deterministic algorithm is able to achieve a sum-squared error near the error threshold with a hidden layer size similar to that for the growth algorithm. However, through observation it is found that in order to achieve the error threshold exactly, a great many more nodes are required. In other works, the asymptotic training behaviour of the deterministic algorithm is not as good as the growth algorithm since the latter possesses the ability of immediately "covering" a "difficult" pattern with its modified nonlinearity while the former has adopted a less radical multiple training pattern node initialization scheme. Therefore, it would be suitable if we can derive a modified convergence criterion in which the error threshold is slightly raised. This also explains the longer training time of the current algorithm.

As usual, the training time decreases when the error gradient threshold increases and is greater than that for the conventional BP algorithm due to the initial single-node network.

Next we would like to compare the recognition rate of the algorithm on both the training set and the test set.

Error Gradient Threshold	Deterministic Algorithm		BP Algorithm	
	Recognition Rate (Training Set)	Recognition Rate (Test Set)	Recognition Rate (Training Set)	Recognition Rate (Test Set)
0.05%	97.8%	78.3%	92.2%	86.7%
0.06%	97.2%	78.9%	90.6%	87.8%
0.07%	96.1%	80.0%	94.4%	87.2%
0.08%	96.1%	81.7%	90.6%	87.8%
0.09%	97.2%	78.9%	94.4%	87.2%
0.10%	96.1%	78.9%	94.4%	87.2%

Table 6.4 The deterministic algorithm applied to the handwritten character recognition problem (recognition rate comparison)

Due to the prolonged training time of the current algorithm as mentioned above, the generalization rate is lower than that of BP and the growth algorithm while higher than that for the PT algorithm. As a result, the advantage of the multi-pattern initialization scheme cannot manifest itself due to the requirement of achieving the error threshold. This would not be the case for the time-series training set as will be seen in the next section. The above conjecture also explains the rather high recognition rate on the training set achieved by the current algorithm as compared to both the BP and the growth algorithm due to the relatively longer training time.

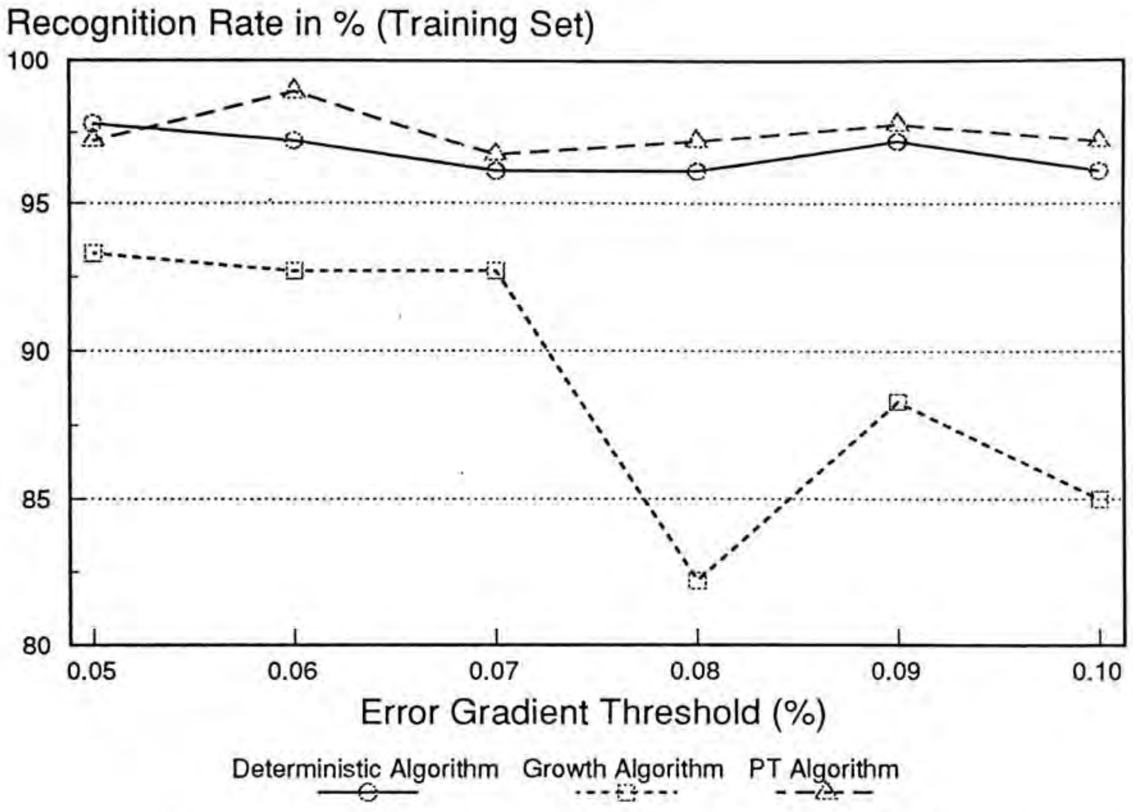

Fig 6.13 Recognition rate (training set) comparison between the three dynamic node creation algorithms (handwritten character recognition problem)

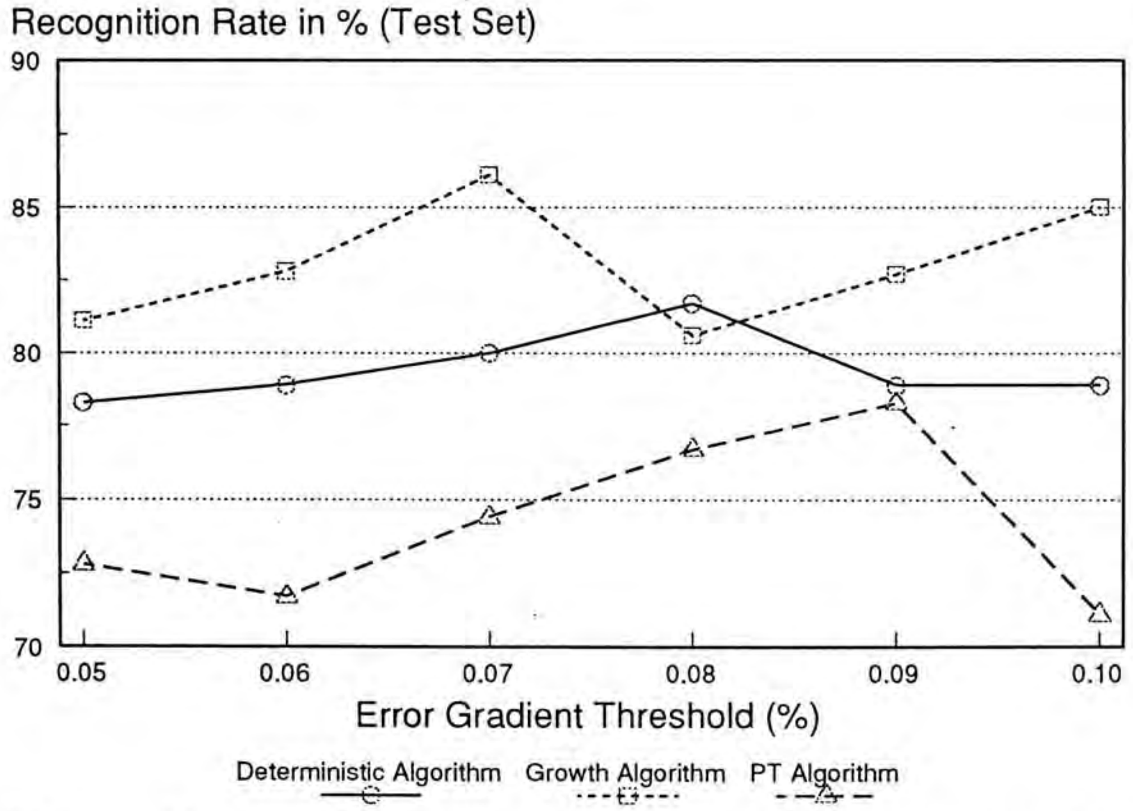

Fig 6.14 Recognition rate (test set) comparison between the two dynamic node creation algorithms (Handwritten character recognition problem)

The generalization capability of the current algorithm also exhibits the characteristic trend of peaking at the intermediate values of error gradient threshold which corresponds to an intermediate training time where either the phenomenon of overfitting or underfitting is not prominent.

3. Time Series Modelling

Finally, the current algorithm is applied to the time series modelling problem using the same training parameters and convergence criterion as in Chapter 4. The results of the various simulation studies is given below.

Error Gradient Threshold	Deterministic Algorithm		BP Algorithm	
	No. of Epochs	No. of Nodes	No. of Epochs	No. of Nodes
0.05%	1520	4	4690	4
0.06%	1457	4	4690	4
0.07%	1413	4	4690	4
0.08%	1365	5	3523	5
0.09%	1337	5	3523	5
0.10%	1276	6	2867	6

Table 6.5 The deterministic algorithm applied to the time series modelling problem (Training speed and network size comparison)

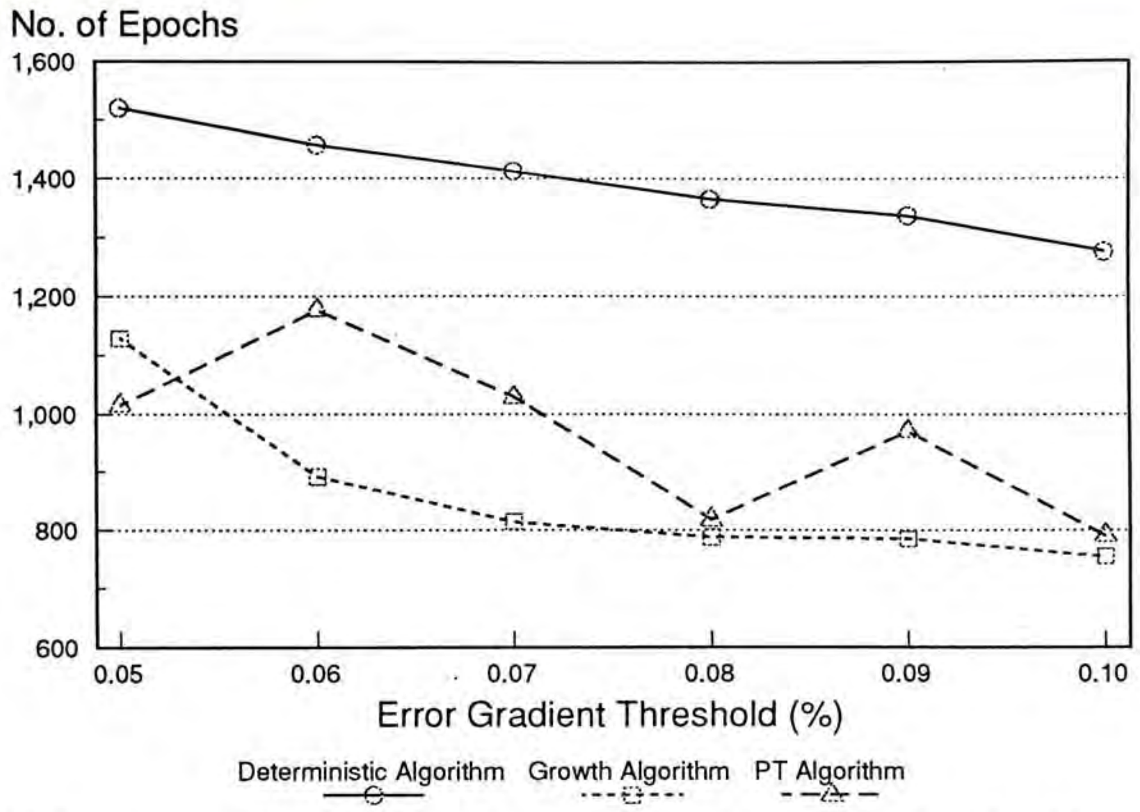

Fig 6.15 Speed comparison between the three dynamic node creation algorithms (Time series modelling problem)

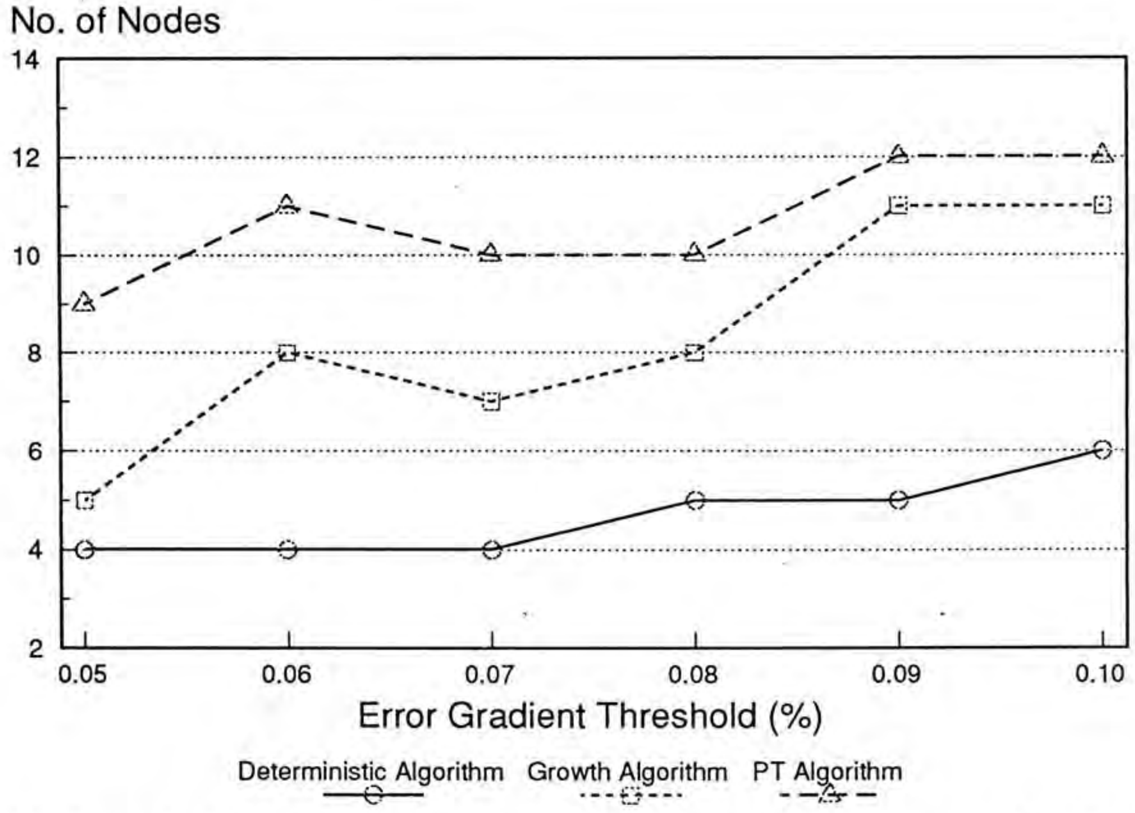

Fig 6.16 Network size comparison between the three dynamic node creation algorithms (Time series modelling problem)

It has been observed in Chapter 4 that the time series training set serves as a difficult problem for the conventional BP network. This phenomenon is again manifested in the current training exercise: the convergence speed of the BP network using the same number of hidden nodes as the current algorithm is in general much lower than that for the current deterministic algorithm. This situation is also much accentuated by the fact that the current algorithm generates a very small resulting network, in fact much smaller than those generated by the growth algorithm and the progressive training algorithm. As a result the current algorithm serves as a better estimator for the network size than for the previous 2 algorithms, while the corresponding small network size used for the BP comparison exercises further accentuate its convergence speed problem.

As expected, the training time required for the current algorithm is longer than that for the growth algorithm due to the special nature of the hidden node initialization scheme of the latter. In this case the progressive training algorithm also outperforms the current algorithm in terms of training speed, but the resulting network size generated by the progressive training algorithm is much larger than that generated by the current algorithm

Next we would like to compare the generalization capability of the various dynamic node creation algorithms:

Error Gradient Threshold	NRMSE (Deterministic Algorithm)	NRMSE (BP Algorithm)
0.05%	0.1033	0.1074
0.06%	0.1032	0.1074
0.07%	0.1031	0.1074
0.08%	0.1041	0.1010
0.09%	0.1052	0.1010
0.10%	0.1090	0.1008

Table 6.6 The deterministic algorithm applied to the time series modelling problem (NRMSE comparison)

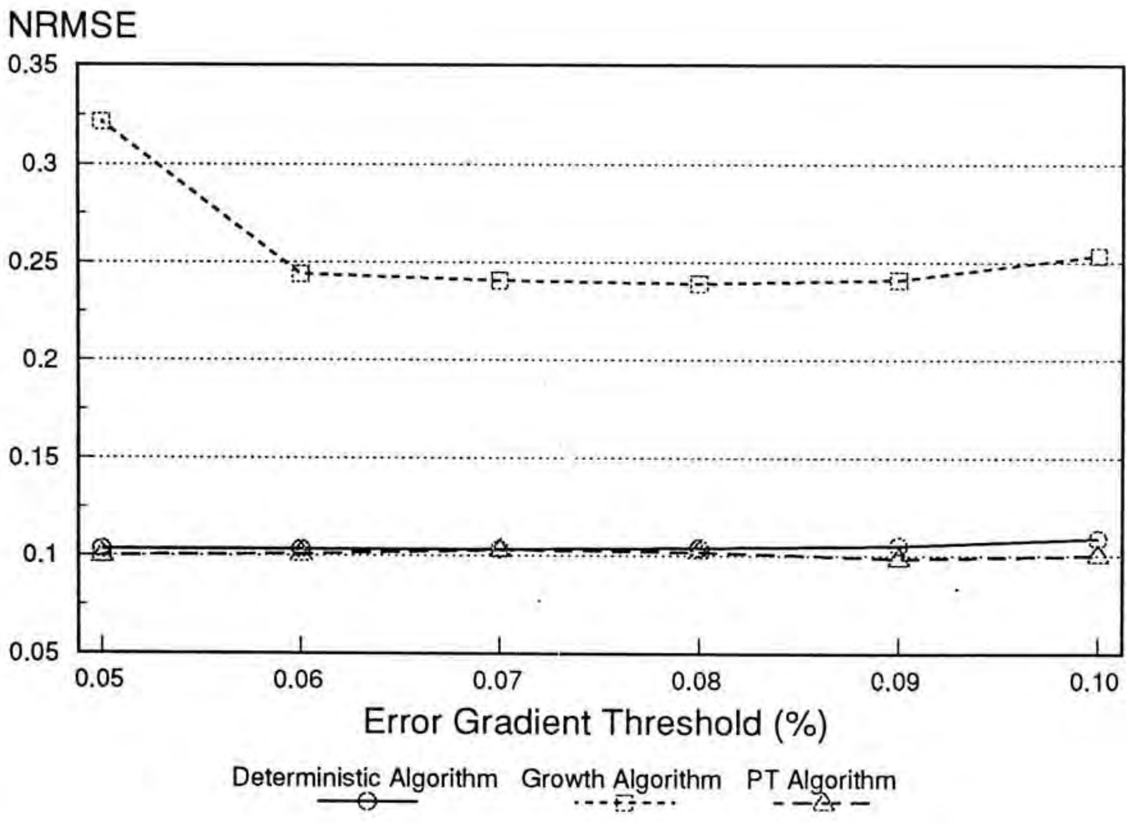

Fig 6.17 NRMSE comparison between the three dynamic node creation algorithms (Time series modelling problem)

It is seen that the generalization capability of the current algorithm is much higher that for the growth algorithm for this particular noisy training set. Thus the advantage of the multi-pattern node initialization scheme has manifested itself in this case when compared with the single pattern initialization scheme of the growth algorithm. In general, the generalization capability of the current algorithm is almost on an equal footing with the BP algorithm and the progressive training algorithm. The simulation results for the current algorithm also exhibits the characteristic trend of peaking at the intermediate range of error gradient threshold values.

In general, the current deterministic dynamic node creation algorithm can successfully cater for the three training sets described in this thesis. The convergence speed for the current algorithm is in general lower than that of the growth algorithm since the node initialization scheme of the deterministic algorithm is not designed to directly "cover" a "difficult" pattern. On the other hand, the merit of the new algorithm lies in its improved new hidden node initialization scheme which makes use of multiple training patterns and thus would give a more accurate estimate for the underlying mapping of the training set, which is indicated by the resulting smaller network size generated by the current algorithm. Moreover, this node initialization strategy discourages the memorization of a single noisy training pattern, which is manifested ins its high generalization capability on encountering noisy data sets such as time series samples when compared with algorithms employing single-pattern initialization such as the growth algorithm. However, the generalization capability of the current algorithm still lags behind that of the BP network due to the early consolidation of the knowledge of the training set in an initial small network. In Chapter 7, the approach of monitoring the generalization measure proposed by Drucker et.al [11] is derived to counteract this problem. In addition, the initial single node network for the current algorithm has manifested itself in the long training time when compared with conventional BP network which starts with multiple hidden nodes. In Chapter 8, an initial hidden layer size estimation procedure is derived such that the current algorithm can start with an initial multiple hidden node network instead of a single node network.

6.11 Concluding Remarks

We have achieved the next step in our course of algorithm development as promised in Chapter 1, in that multiple training patterns can now be utilized to initialize a new hidden node in a dynamic node creation environment, thus avoiding once and for all the possibility of the memorization of noisy training data by the new hidden node, as would be possible under the progressive training algorithm and the growth algorithm. Moreover, the current algorithm do not assume any specific features either in the training sequence or in the training set organisation (as in progressive training) or in the architecture of the network (as in the growth algorithm). The conception of the current algorithm is made possible through the knowledge representation model developed in Chapter 5, in which the powerful T-vector approach help us to elucidate more details concerning the relationship between the hidden layer and the output layer in which the conventional S-vector approach is unable to provide. The current algorithm is successfully applied to the parity problem, a handwritten character recognition problem, and a time series prediction problem. The resulting network generated by the current algorithm all compares favourably with that generated by the progressive training algorithm and the growth algorithm in terms of both the network size and training speed. In addition, the algorithm generates a near optimal network size for all parity orders in the training of the parity data set. However, it is observed over the past few chapters we have noticed that networks created by dynamic node creation algorithm generally exhibit lower generalization rate than their BP counterparts, and the training speed is low whenever the training set is simple enough for the conventional BP algorithm such that BP has a headstart with its initial multi hidden node network. In Chapter 7 we explore the possibility of using the generalization measure derived by Drucker et.al [11] as a node addition criterion. In addition, we explore the possibility of generating a multi hidden node initial network for the present algorithm in contrast to the initial single node network in Chapter 8.

7 THE GENERALIZATION MEASURE MONITORING SCHEME

7.1 The Problem of Generalization for Neural Networks

Through the development of the BP training process, the artificial neural network has become a truly practical mapping device in that a training set can be constructed from the mapping by appropriately sampling the function and produce an adequate number of input-output pairs. The BP process can then accept these input-output pairs (or training patterns) and produce a neural network with the appropriate network weights to implement the desired mapping. But in a strict sense the neural network is not learning the underlying mapping of the training set but only the restricted mapping represented by the training patterns. As a result, if the training pattern is not selected properly such that their characteristics are general enough to represent the underlying mapping of the training set, the mapping obtained by the neural network by loading these training patterns onto the network would differ considerably from the original mapping. This is a severe limitation of data representation by neural network as it is not possible to train a neural network with a training set of infinite size. However, if the underlying mapping of the training set is smooth enough such that the value of the function at a certain point is well-approximated by the function values at neighbouring points, then the training of the network is equivalent to the learning of the underlying mapping of the training set, provided that the training time is not so long as to cause the cohesion of the network mapping to the sample points of the training set, which result in an overall "wrinkled" surface instead of the smooth surface we have desired. The generalization capability of the network would not be severely disturbed under this situation, but it is almost certain that the generalization capability under this prolonged training condition would be lower than the case where the training time is suitably adjusted such that the mapping retains its smooth appearance. The problem is: how can we define this so called suitable training time? We may attempt to train the network for a "not so long" period, but since we have no guideline in selecting this training time, the approximation error on the training set may worsen due to the shortened training time. Any attempt in deciding this training time through the monitoring of the mean square error of the network is also futile as there seems no clue as to which mean square error exhibited by the network would lead to the best generalization capability for the network, except that an overly small mean square error value would definitely lead to a worsened generalization capability. It is obvious from the above consideration that we must rely on additional information in solving the generalization problem, as the information given by the training set does not prepare the neural network for the infinite varieties of test set that it would meet.

7.2 Prior Attempts in Solving the Generalization Problem

Traditionally, it is believed that a network with a smaller number of hidden nodes will generalize better than a network with a larger number of hidden nodes, since the smaller number of parameters in a small network forces the training algorithm to generalize over the training patterns instead of memorizing each pattern individually , as this is the only alternative for the training algorithm to produce an acceptable mean square error at the network output since it simply does not have enough resources. On the other hand, for a network with a large number of hidden nodes, the network will tend to memorize some of the training patterns due to the large number of training parameters available. Some researchers [52] suggested that the reduced computational power of a small network can be compensated by using more hidden layers. The increased number of parameters would not affect the generalization capability of the network as the information in the training set has already been squeezed into a compact code at the lower layer to be interpreted at the higher layer, which is different from the situation where the same number of parameters is present in a single hidden layer. Adhering to this principle of minimum network size, Rumelhart et. al [58] has derived a pruning procedure for the hidden layer which attempts to remove redundant nodes in a network with the hope that the resulting smaller network will perform a better task of generalization. They achieved this by including an additional term in the BP cost function which encourages the rapid decay of network weights with small magnitudes, thus hidden nodes which play minor roles in the approximation of the training set will gradually wither away. Rumelhart applied this algorithm to the task of time series prediction in which the training patterns are especially susceptible to noise and found that the pruning process has actually improved the prediction capability of the network. They have also proposed an alternative method in which a portion of the training set is chosen as a so-called validation set and does not participate in the training process. During the training process, the mean square error invoked at the training patterns in the validation set are monitored, and when the mean square error at these points reaches a minimum and starts to rise, the training process is terminated. It was found that this process can also improve the generalization capability of the network. In general, the two methods have their respective disadvantages: for the first method , delicate balance has to be achieved between the two terms in the cost function to achieve the optimum effect. For the second method, we have to sacrifice a portion of the training patterns which should have been used to decrease the mean square error of the network. The above methods, in particular the pruning approaches, are based on the notion that small networks tend to generalize better. However, the notion is found to be invalid for some training sets. For example, Siestma et. al [52] discovered that, for networks trained with noisy inputs, large networks actually generalize better than small networks, thus indicating that the above notion cannot be applied universally. An alternative method of improving the generalization capability of a network is to generate additional training patterns from the original training patterns by adding noises to the original patterns. The intention of the above practice is clear: besides requiring the network to attain the desired output at a designated training pattern, we would also require that patterns at the vicinity of the chosen training pattern should adopt a more or less similar network output. Interpreted in another context, we are trying to anticipate the possible patterns in the test set by disturbing in different ways the training patterns which are available to us. Matsuoka [33] proved mathematically that the adoption of this noise injection approach would actually reduce the sensitivity of the network to variation in the input, which is another way of saying that the generalization capability of the network is increased. Holmstrom et. al [22] provided guidelines for selecting the probability density of the noises to be added to the training patterns if this approach is adopted in improving the generalization capability of the network. Recently, a completely new approach is derived to address this problem, which is to be described in the next section.

7.3 The Generalization Measure

The generalization measure derived by Drucker and Le Cun [11] represents a completely new way of addressing the problem of generalization. The derivation of this method is based on the notion that a network mapping which serves as a good model of generalization should be relatively smooth and free of abrupt transitions. This in turn implies that the derivative of the error function with respect to the input patterns should be small, such that the network error will not change much when the input patterns are changed slightly, as will be the case for most patterns in the test set. Drucker et. al simply adopted this derivative as part of the BP cost function such that in the BP training process, this quantity is minimized alongside the mean square error function in a way such that the generalization capability and mean square performance of the network. Formally, this additional term E_b for the cost function is given as

$$E_b=\frac{1}{2}\sum_{i=1}^{n}(\frac{\partial E}{\partial x_i})^2 \qquad (7.1)$$

Where the x_i's, i=1 to n, are the various network inputs and E is the normal sum-squared error function. This derivative, multiplied by the generalization coefficient α, is added to the original sum-squared error function to produce the total error function E_t

$$E_t=E+\alpha E_b \qquad (7.2)$$

We would hereafter call this total error function the generalization measure of the network. Drucker et. al has evaluated the derivative of E_t with respect to all weights in the network such that a BP-type algorithm can be applied using the total error function E_t instead of the usual mean square error function E. Since the calculation; of all the derivatives involve a first pass through the original network and a second pass through a so called appended network which helps in calculating the derivative of E_b with respect to all weights, the new learning algorithm is referred to by the authors as the double backpropagation algorithm. As can be seen from the above description, the evaluation of the derivative of E_t with respect to all weights in the network can be a complicated process as a double pass through the network is required. However, the calculation of E_b is a simple task as all the quantities required for its calculation is available from the original BP process: in the original BP process a quantity δ_j, j=1 to q, is defined for each hidden node. We can backpropagate these error terms through one more step to the input layer using the original BP rule to obtain each of the terms in E_b:

$$\frac{\partial E}{\partial x_i}=\sum_{j=1}^{q} w_{ij}\delta_j \qquad i=1,....,n \tag{7.3}$$

With the availability of these partial derivatives, we can calculate E_b by squaring each derivative and summing together.

7.4 The Adoption of the Generalization Measure to the Deterministic Algorithm

We have mentioned the calculation of E_b only, since this is the only calculation procedure required for adopting the concept of generalization measure to the deterministic algorithm. The motivation for the adoption of this approach to the deterministic algorithm is due to the observation that the generalization capability of the network created by the present algorithm is in general not as high as a network created by the conventional BP process. For some training set (especially the handwritten character recognition set), this phenomenon is not restricted to the deterministic algorithm but is observed whenever a network is created through the dynamic node creation method, as can be seen in the case of the progressive training algorithm and the growth algorithm. This may be due to the early consolidation of the knowledge embedded in the training set in a small network which the further training step is unable to eradicate. Since a small network is in general not adequate to represent most training sets, misrepresentation of the training set will result and leads to a poor generalization rate. This problem can be alleviated if the error gradient threshold G, which is an essential parameter in deciding when to add new nodes to the network, is raised such that the training set knowledge is not so deeply embedded in the early small network. This approach, however, will in general lead to a large network due to the relaxation of the node addition criterion and in turn lead to a waste of resources. It would be ideal if the error gradient threshold G can vary suitably in each instance of node addition such that a balance can be struck between generalization capability and the network size. Here we suggest the monitoring of the previous generalization measure throughout the whole BP fine-tuning process on the addition of a new node to find a suitable moment to terminate the training. If the generalization measure really corresponds to the generalization capability of the network, then the presence of the generalization measure is equivalent to the presence of a validation set in the sense defined by Rumelhart et. al [58]. However, in this case, we have the added advantage that we do not have to sacrifice any training patterns in the training set to create the validation set. The correspondence between the generalization measure and the generalization capability of the network will be studied in section 7.6. In the next section, we would first describe our proposed method of monitoring the generalization measure.

7.5 Monitoring of the Generalization Measure

Through experimental studies, it is observed that on addition of a new hidden node and the application of the BP fine-tuning process, the generalization measure will assume the shape as depicted in Fig 7.1.

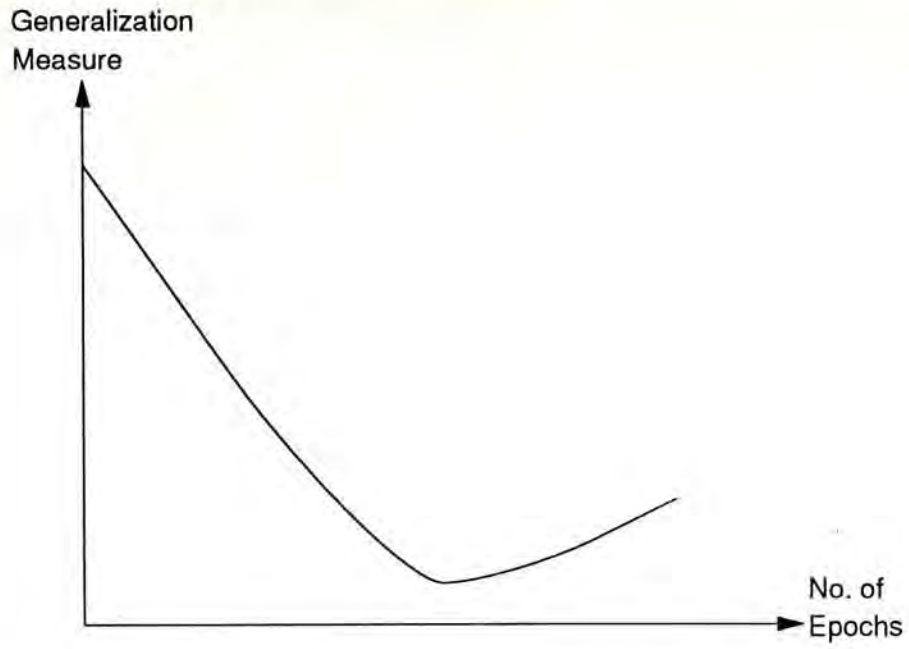

Fig 7.1 The generalization measure profile

Corresponding to the approach of Rumelhart et. al [58] on the validation set, we would terminate the training whenever the generalization measure exhibit its deepest trough. This can be equivalently achieved by the following series of steps:

(1) Set $E_{t, min}$ to a large number .

(2) Calculate $E_t = E_t(n) - E_t(n - 1)$ at each step of the BP fine-tuning process.

(3) Whenever $E_t > 0$, compare $E_{t, min}$ with the current E_t. If E_t is smaller than $E_{t, min}$, record the current state of the network and replace $E_{t, min}$ with the current value of E_t.

(4) At the end of the BP fine-tuning process when $\Delta E/E$ is smaller than its threshold, compare the current E_t with $E_{t, min}$. If E_t is larger than $E_{t, min}$, replace the current state of the network with the stored state, otherwise retain the current state of the network.

The above process constitutes a simple process of monitoring the generalization measure. In addition, since the generalization measure can be calculated using quantities which are already available through the standard BP process, the monitoring of this additional quantity does not constitute a substantial overhead in the implementation of the deterministic algorithm and thus serves as a practical method for improving the generalization capability of the network under a dynamic node creation environment.

7.6 Correspondence between the Generalization Measure and the Generalization Capability of the Network

The variation of the generalization measure and the generalization capability of the network throughout a certain BP fine-tuning process is depicted in Fig 7.2 to Fig 7.7. The first three figures correspond to the handwritten character recognition problem and the last three figures correspond to the time series modelling problem. For each training set, the three figures correspond to α=1, 2 and 3.

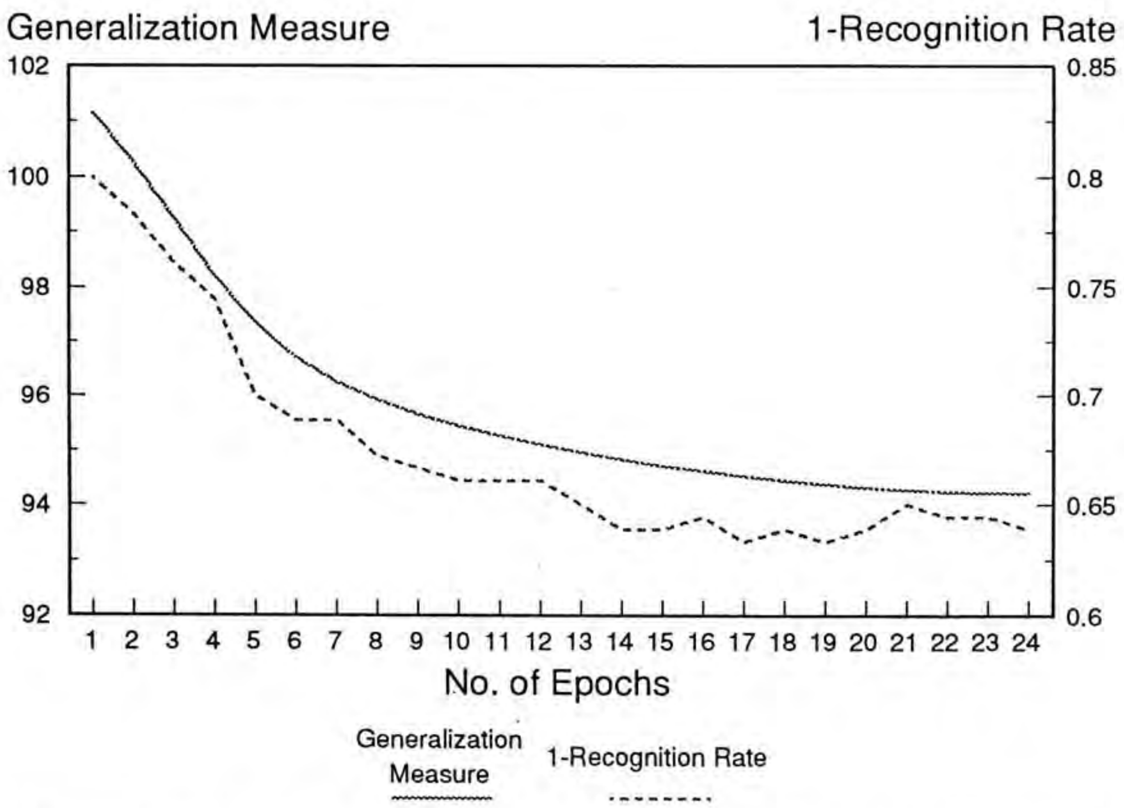

Fig 7.2 Generalization measure profile of the handwritten character recognition problem (α=1)

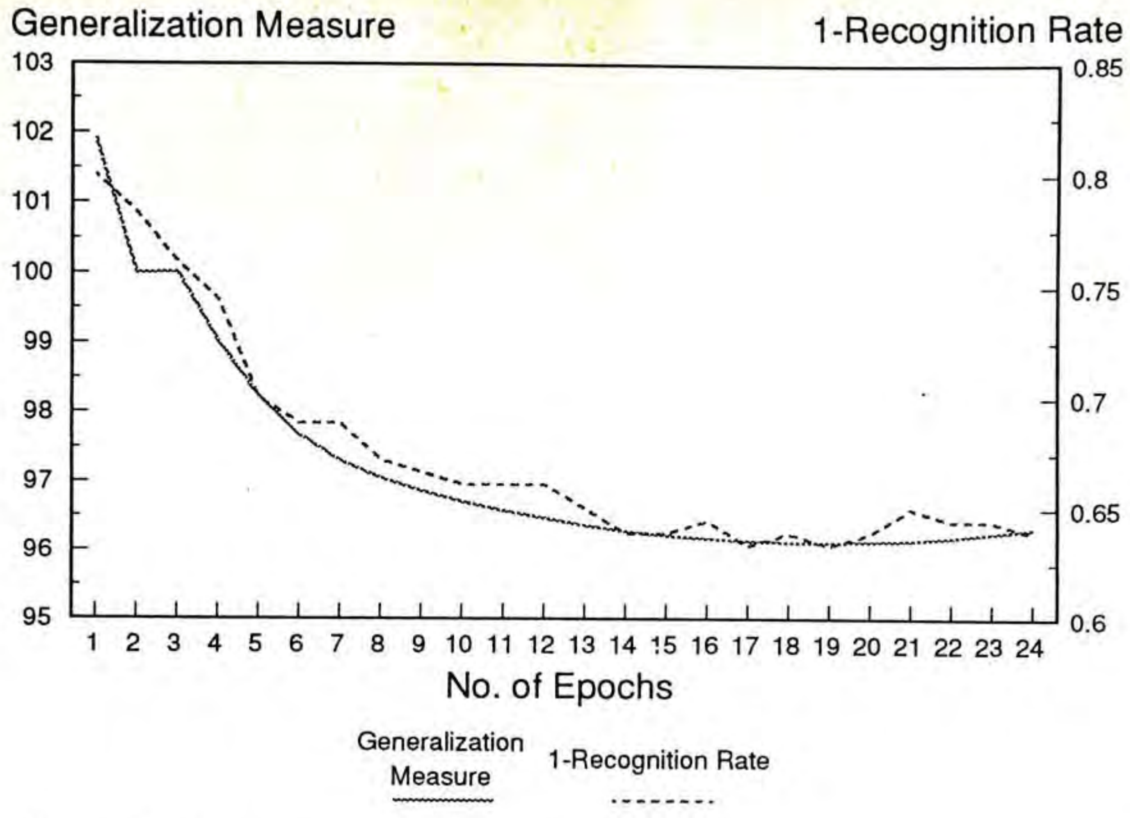

Fig 7.3 Generalization measure of the handwritten character recognition problem (α=2)

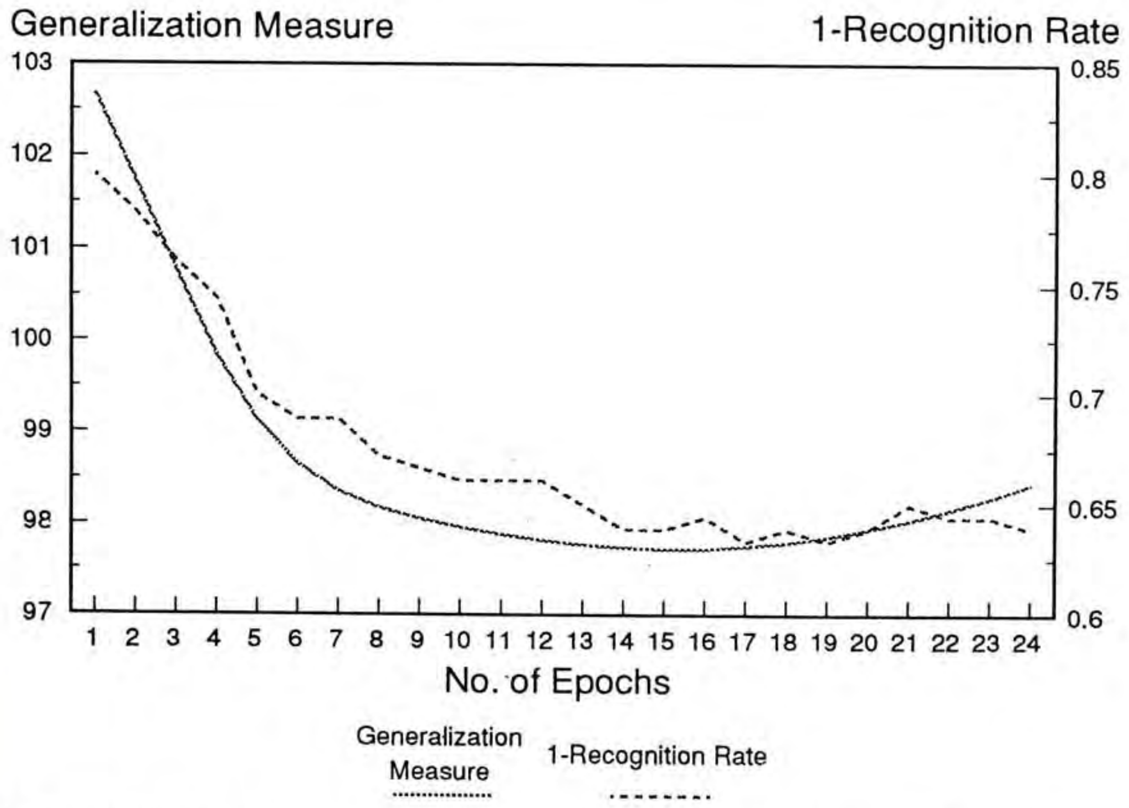

Fig 7.4 Generalization measure for the handwritten character recognition problem (α=3)

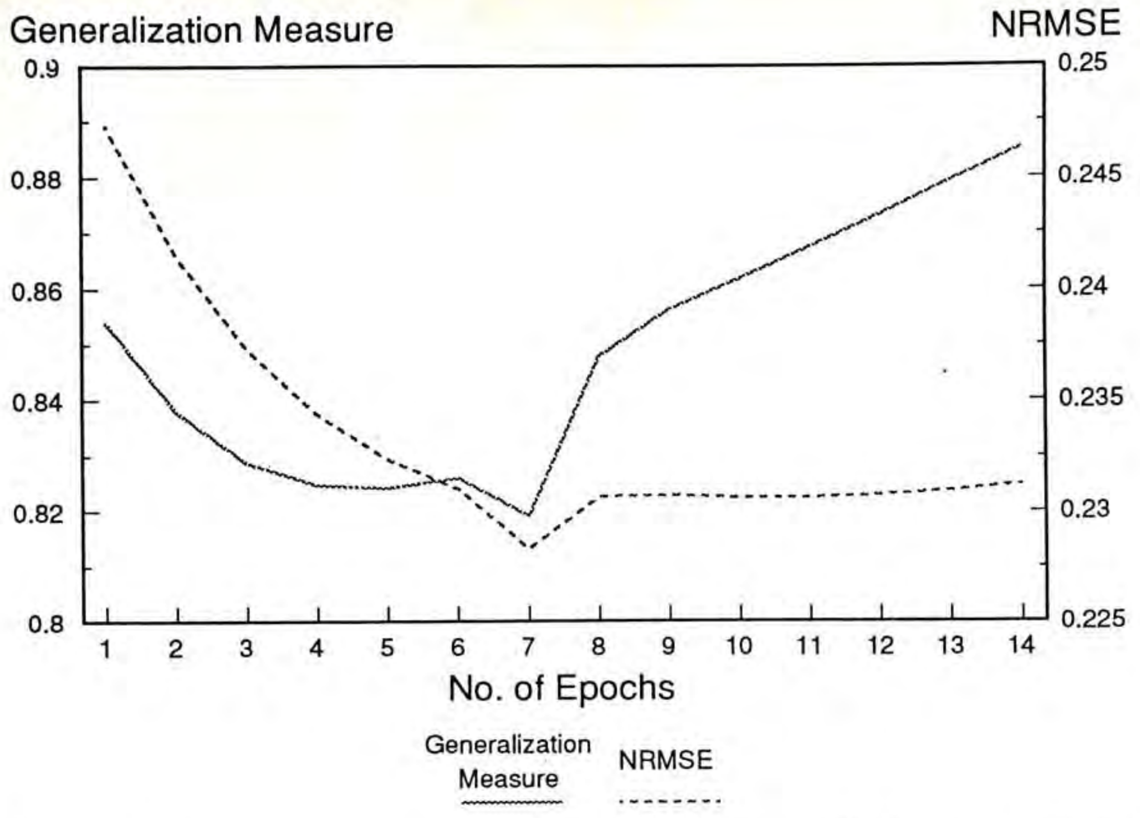

Fig 7.5 Generalization measure profile of the time series modelling problem (α=1)

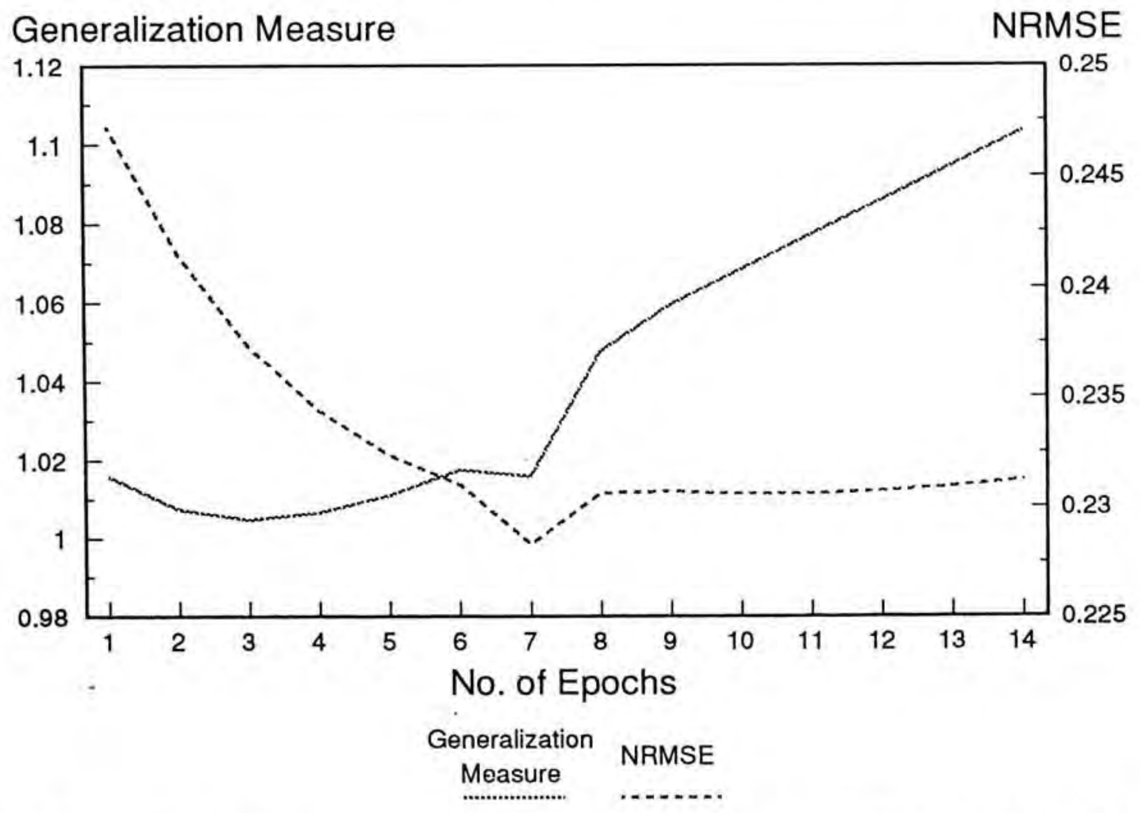

Fig 7.6 Generalization measure profile of the time series modelling problem (α=2)

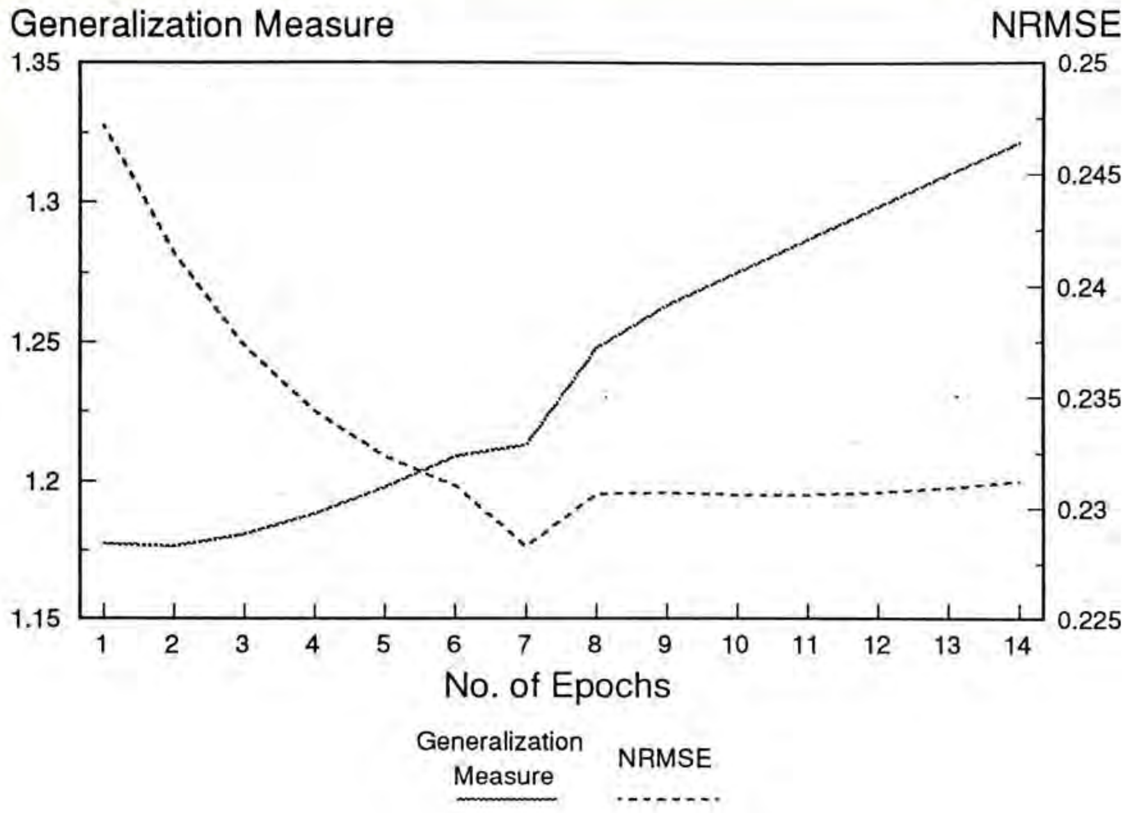

Fig 7.7 Generalization measure profile of the time series modelling problem (α=3)

For investigating the generalization measure profile of the handwritten character recognition problem, we have made use of the quantity (1-Recognition Rate) on the test set, since this quantity decreases with the generalization capability of the network which corresponds to the behaviour of the generalization measure, such that an effective comparison can be made between these two quantities. In Fig 7.2, 7.3 and 7.4, which corresponds to α=1, 2 and 3, we can observe that the generalization measure correlates highly with the generalization capability of the network in that both decreases almost in a monotonic way during almost the entire course of training while exhibiting a slight upturn at the end of training which indicates the beginning of deterioration of the network's generalization capability. Thus the generalization measure can replace the validation set as an effective indicator of the network's generalization capability. For α=3, the upturn for the generalization measure occurs earlier due to the more important role played by the input derivative terms in the overall error function. Thus a large value of α will lead to a more conservative estimate of the overall generalization situation and the generalization measure monitoring scheme would attempt to restore the network to a state before the deterioration of the generalization capability. Therefore, from the above empirical observations we can conclude that $\alpha=2$ serves as a suitable parameter for the generalization measure monitoring scheme.

In Fig 7.4, 7.5 and 7.6, the same phenomenon is observed for the time series modelling problem in that the same correspondence between the generalization measure and NRMSE is found. The NRMSE at first decreases in a monotonic way and then starts to increase due to the overfitting of the training set. The generalization measure exhibits a similar trend by starting to increase at a moment which depends on the value of α. In general, the upturn for $\alpha=1$ occurs at a later period and that for $\alpha=3$ occurs at an earlier period which constitutes a conservative estimate of the overall generalization situation. Therefore, we can conclude that the parameter $\alpha=2$ is again suitable for the current time series modelling problem.

7.7 Experimental Results and Performance Analysis

In [11], the authors suggested that the value of the generalization coefficient α should be greater than unity. In the simulation studies, we have attempted the values of $\alpha=1$, 1.5, 2, 2.5, 3 and to observe the subsequent generalization behaviour of the network. The lowest error gradient threshold value of 0.05% is chosen for the simulation studies in order that a wider range of the variations of the generalization measure is experienced. We have tested this generalization measure monitoring scheme on the handwritten character recognition problem and the time series modelling problem.

(1) The Handwritten Character recognition Problem

The results of applying the scheme to the character recognition problem is given below:

Generalization coefficient α	No. of epochs	No. of nodes	Recognition rate (training set)	Recognition rate (test set)
1.0	420	25	97.8%	79.4%
1.5	389	21	96.7%	78.9%
2.0	288	22	98.3%	85.0%
2.5	269	16	96.7%	86.1%
3.0	193	18	96.7%	83.9%

Table 7.1 The generalization measure monitoring scheme as applied to the handwritten character recognition problem

Error Gradient Threshold	No. of Epochs	No. of Nodes	Recognition rate (training set)	Recognition rate (test set)
0.05%	1189	30	97.8%	78.3%
0.06%	1365	26	97.2%	78.9%
0.07%	960	27	96.1%	80.0%
0.08%	850	26	96.1%	81.7%
0.09%	818	27	97.2%	78.9%
0.10%	656	27	97.2%	78.9%

Table 7.2 The original deterministic algorithm without the monitoring scheme as applied to the handwritten character recognition problem

From the results it is observed that for $\alpha=1$, the generalization rate is almost the same as that for the case when no monitoring of the generalization measure is applied, since the weighting of the additional term is relatively low. However, it is observed that as α increases, the generalization rate of the network increases accordingly. The rate peaks at $\alpha=2.5$ after which it starts to fall off. As a result, we can conclude that monitoring the generalization measure is an effective means of improving the generalization capability of dynamic node creation algorithm provided that the correct range of α is chosen. The results also confirmed the conjecture of Drucker et. al [11] that the value of α should be much greater than 1 in order for the measure to mirror correctly the generalization capability of the network. Moreover, it is observed that the convergence speed is greater than that of the unenhanced scheme. This can be explained by the realization that the current monitoring scheme would in general restore the network to an earlier stage of training in the BP fine-tuning process than the unenhanced scheme. As a result, the monitoring scheme is equivalent to the selection of a higher error gradient threshold and thus the greater convergence speed for the network.

(2) Time Series Modelling

The results for applying this scheme to the time series modelling problem is given below:

Generalization coefficient α	No. of epochs	No. of nodes	NRMSE
1.0	1555	4	0.1033
1.5	1534	9	0.1027
2.0	852	10	0.1024
2.5	772	6	0.1106
3.0	863	7	0.1099

Table 7.3 The generalization measure as applied to the time series modelling problem

Error Gradient Threshold	No. of epochs	No. of nodes	NRMSE
0.05%	1520	4	0.1033
0.06%	1457	4	0.1032
0.07%	1413	4	0.1031
0.08%	1365	5	0.1041
0.09%	1337	5	0.1052
0.10%	1276	6	0.1090

Table 7.4 The original deterministic algorithm without the monitoring scheme as applied to the time series modelling problem

The results for the training of the Mackay Glass time series is similar to that of the previous problem. In this case, the generalization capability of the network peaks at

$\alpha=2$. However, the improvement in the generalization capability is not dramatic as the generalization capability of the unenhanced scheme for this problem is already very high. However, in view of the fact that the enhanced scheme with the optimum α still provides a lower NRMSE than the unenhanced scheme and there is substantial improvement in the generalization capability for some other training sets (e.g. the character recognition training set) , it is still worthy to implement the enhanced monitoring scheme.

Summarizing the experience of the previous two simulation studies, we can conclude that the value of α should be approximately between 2 and 2.5, since too low a value of α would result in only a scant improvement in the overall generalization capability of the network, while too large a value of α would on the contrary decrease the generalization performance of the network. Through the simulation studies, it is observed that , though the generalization performance of the network has improved, the convergence speed is still low when compared with the conventional BP approach. In chapter 8, an initial hidden layer size estimation procedure would be derived to counteract this problem.

7.8 Concluding Remarks

In this Chapter we have derived a method of monitoring the generalization measure proposed by Drucker et. al [11] in order to improve the generalization capability of networks built up under the dynamic node creation environment. The utilization of this approach is equivalent to the adoption of an adaptive $\Delta E/E$ threshold at each stage of node addition instead of adopting a fixed threshold, which helps to prevent the early consolidation of the training set into a small network. We have applied this procedure to the handwritten character recognition data set and the time series data set. We have found that the generalization capability of the network has actually increased in some instances, especially for those cases when the generalization capability discrepancy between a network created using node addition methods and a network created using BP is large. We have observed in Chapter 6 that the networks created by these two methods are not interchangeable as can be observed from the different generalization capability of the two network when the same terminating error threshold is applied to both of the networks. The suggested monitoring scheme would serve to narrow this gap by first attempting to narrow their generalization gap such that a dynamic node creation algorithm can eventually produce a network of the same quality as that created by BP in order to pave the way for the replacement of BP with the former learning strategy.

8 THE ESTIMATION OF THE INITIAL HIDDEN LAYER SIZE

8.1 The Need for an Initial Hidden Layer Size Estimation

A common characteristic for dynamic node creation algorithm is that the process must start with a single node network, as for the cases in the progressive training algorithm, the growth algorithm and the deterministic algorithm. The reason for this initial state of the network is that if we start with any other network configuration other than a single-node network, there is the possibility that the initial network may already be more than adequate in solving the problem presented by the training set, which leads to a waste of resources. Therefore, we should start with as few hidden nodes as possible, and sinces a single-node network is the barest minimum of all networks which contains a hidden layer, we should adopt this configuration as our initial network state. The advantage of this approach is that the possibility of the inclusion of redundant nodes in the network is greatly reduced. The disadvantage of this approach is that the training time would be greatly prolonged as can be seen from the experimental results in Chapter 6. This situation becomes more and more critical when the eventual network size which is capable of representing the training set is large, which translates into a great many number of node addition procedures for the dynamic node creation algorithm. The situation is further worsened when the problem would be simple once when the adequate number of hidden nodes are provided, but would otherwise by very difficult. The disparity between the two modes of training under this situation would be further widened.

The problem stated above seems to be an unsolvable one since we really cannot estimate the hidden layer size of the network, or there would be no need for the derivation of dynamic node creation algorithm! However, it would still be possible that an approximate initial number of hidden nodes is estimated for the network before applying the dynamic node creation algorithm, provided that we are willing to tolerate the existence of a few redundant nodes: an underestimation of the initial hidden layer size would not pose too much a problem as the subsequent node addition procedure would attempt to fill in the missing nodes. However, there would be the possibility that the initial hidden layer size would be an overestimation, in which case redundant nodes would be included in the network. Therefore, an ideal initial hidden layer estimation scheme should attempt to minimize the occasions of overestimation.

We would now suggest an initial hidden layer estimation scheme for the deterministic dynamic node creation algorithm proposed in Chapter 6. This scheme is based on the notion that the hidden T-vector will become one of the neural basis $\mathbf{n}_j$ introduced in Chapter 6 when the hidden weights of the node is large enough. This scheme is described in the next section.

8.2 The Initial Hidden Layer Estimation Scheme

Based on the notion that each hidden node will eventually emit one of the neural bases $\mathbf{n}_j$, we could visualize the estimation of the initial hidden layer size as the selection of the appropriate neural basis to be included in the initial collection of hidden T-vectors. For an arbitrary training set containing p patterns, in general the full set of neural basis $\mathbf{n}_j$ j=1 to p, would be required to exactly represent the training set. However, it is observed through experimental studies that a neural network is capable of adequately representing a training set using far fewer hidden nodes than the case when the full set of neural basis are required. The reason for this small hidden layer size is that the inverse desired output T-vector $\mathbf{d}_1^{-1}$ (assuming a single-output network) is closer to a subset of the complete set of neural bases such that the former can already be well approximated by the latter when only this subset of neural bases is present in the neural network. This situation is depicted in fig 8.1.

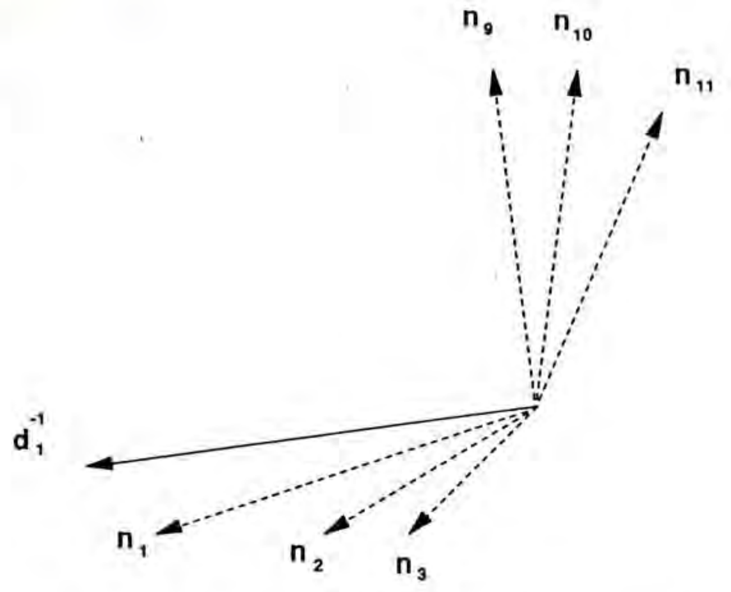

Fig 8.1 The relationship between $\mathbf{d}_I^{-1}$ and the neural basis

From Fig 8.1 It is seen that it is not necessary for the inverse desired output $\mathbf{d}_I^{-1}$ to be equidistant from all the neural basis, and that it may preferentially incline towards a special subset of the neural basis and being far away from all the other basis in the complete basis set. Although the notion that the hidden nodes are emitting only the neural basis must be based on the assumption that the underlying hidden weights of the node must be relatively large, which may not be true in practice, this picture of the relationship between $\mathbf{d}_I^{-1}$ and the neural basis at least provides a possible channel for the estimation of the initial hidden layer size of the network through this initial assignment. Moreover, even when the hidden weights of a node is not sufficiently large, the hidden T-vector would still somewhat resemble the underlying neural basis of the node as the same orientation of the hyperplane of that hidden node would correspond to one of the neural basis when the hidden weights become large. In fact, after we have determined the initial hidden layer size, by selecting the appropriate subset of neural basis, we would not initialize each of the nodes with the exact neural basis by assigning an arbitrary large values of hidden weights, but would instead adopt a relaxed version of initialization, as for example, using Eq (3.1) when the scale-up factor is small. This process is followed by a BP fine-tuning process such that the correct weights of the network can be estimated.

The detailed procedure for the estimation of the initial hidden layer size will now be described. Assuming a single-output network with a single inverse desired output T-vector $\mathbf{d}_1^{-1}$ and defining the operator ang(**x**, **y**) as the cosine of the angle between the two vector **x** and **y**, i.e.

$$ang(x,y)=\frac{<x,y>}{\|x\|\cdot\|y\|} \tag{8.1}$$

The patterns in the training set is first ordered according to their Euclidean distances such that our subsequent work can involve the original neural basis $\mathbf{n}_j$ with its various components unpermuted. This does not constitute a restriction of the ability of the algorithm in performing incremental learning, as only the initial state of the network depends on this ordering process, while the subsequent node addition process does not depend on this process. As a result, new training patterns can still be added to the training set with the confidence that new nodes can always be added to the network to cater for the new training patterns.

We would first proceed to calculate this angle operator between the inverse desired output vector and all the neural basis, i.e. we would calculate

$$s_j=ang(d_1^{-1},n_j) \qquad j=1,....,p \tag{8.2}$$

and selecting the maximum value of s_j as s_{max}

$$s_{max}=\max\{ang(d_1^{-1},n_j)\} \qquad \forall j \tag{8.3}$$

The value of s_j is an indication of the angular distance between $\mathbf{d}_1^{-1}$ and the various $\mathbf{n}_j$. The larger the s_j for a particular $\mathbf{n}_j$,the nearer will $\mathbf{d}_1^{-1}$ be closer to this $\mathbf{n}_j$ in terms of angular distance. The $\mathbf{n}_j$ which corresponds to s_{max} is the neural basis which is closest to the desired inverse output T-vector. Therefore we must select those $\mathbf{n}_j$ with a corresponding large s_j into the initial set of neural basis in the network which we designated as **N**. Unfortunately, the criterion of being "relatively near" to the inverse desired output T-vector is difficult to define: we cannot determine for what range of s_j should the corresponding $\mathbf{n}_j$ be considered relatively close to $\mathbf{d}_I^{-1}$. As a result, we must resort to simulation studies to determine in what magnitude range of s_j should the corresponding $\mathbf{n}_j$ be adopted into the set **N**. The implementation of this process is facilitated by the definition of an additional quantity r_j where

$$r_j = \frac{s_j}{s_{\max}} \qquad \forall j \tag{8.4}$$

The magnitude range of r_j is between 0 and 1 and there is at least one $\mathbf{n}_j$ whose corresponding r_j is 1. We should hereafter place a certain $\mathbf{n}_j$ into the initial neural basis set **N** if the corresponding r_j is greater than a certain threshold T, i.e

$$n_j \in N \quad if \quad r_j > T \tag{8.5}$$

We designate the quantity T as the angular distance threshold. This threshold T should be smaller than 1 such that the initial neural basis set consists of at least one $\mathbf{n}_j$. The decision of the value of T is equivalent to deciding a measure of closeness between $\mathbf{d}_I^{-1}$ and the $\mathbf{n}_j$ and is the one value which we should resort to experiments to decide. A reasonable basis of selection is to choose those T values which are close to one. In the simulation studies we have assigned the values of 0.95, 0.9, 0.85, 0.8 and 0.75 for the T values and compared the size of the network generated by the present estimation procedure and the deterministic dynamic node creation procedure in order to decide on a suitable T for further works.

With the determination of the initial neural basis set **N**, we can prepare the same number of initial hidden nodes as the number of neural basis in **N**, and initialize each hidden node according to Eq (3.1) with the scale-up factor s equal to 1, i.e. if $\mathbf{n}_j$ belongs to the set **N**, then Eq (3.1) is applied to one of the initial hidden node using the j-th pattern $\mathbf{x}(j)$.

The output weights of the network is determined in the usual way using Eq (6.19) and from then on the present initial hidden layer size procedure can then be coupled to the former deterministic dynamic node creation algorithm.

8.3 The Extension of the Estimation Procedure to the Multi-Output Network

Instead of using the original inverse desired output T-vectors $\mathbf{d}_l^{-1}$ k=1 to m in the multi-output network, we use the converted basis $\mathbf{c}_k^{-1}$ k=1 to m which span the same space $\mathbf{D}^{-1}$ and which are much closer to the input space **X**. The estimation procedure is applied to each of the converted basis $\mathbf{c}_k^{-1}$ which results in m sets of initial neural basis set $\mathbf{N}_1,\ldots..\mathbf{N}_m$. We suggest that the initial neural basis set for the whole network should be given by

$$N=N_1 \cup N_2 \cup \ldots\ldots \cup N_m \tag{8.6}$$

This assignment method for **N** will ensure that the resulting network will cater sufficiently for all the converted bases $\mathbf{c}_k^{-1}$.

8.4 Experimental Results and Performance Analysis

(1) The Parity Problem

The initial hidden layer estimation scheme, when applied to the parity problem gives the following results. The training parameters for the dynamic node creation portion of the algorithm are the same as those used for the same problem in Chapter 6:

Parity Order	angular distance threshold r_t	Initial hidden layer size (Enhanced algorithm)	Final hidden layer size (Enhanced algorithm)	Hidden Layer Size (Original Algorithm)
7	0.95	1	11	7
	0.90	3	11	
	0.85	5	9	
	0.80	8	9	
	0.75	12	13	
6	0.95	1	9	6
	0.90	2	10	
	0.85	4	7	
	0.80	5	8	
	0.75	8	9	
5	0.95	1	10	7
	0.90	2	5	
	0.85	2	5	
	0.80	3	6	
	0.75	4	6	

Table 8.1 Initial hidden layer size estimation procedure applied to the parity problem (parity order 5 to 7, hidden layer size comparison)

Parity order	angular distance threshold r_t	Initial hidden layer size (Enhanced algorithm)	Final hidden layer size (Enhanced algorithm)	Hidden layer size (Original algorithm)
4	0.95	1	4	5
	0.90	1	4	
	0.85	1	4	
	0.80	1	4	
	0.75	1	4	
3	0.95	2	3	3
	0.90	2	3	
	0.85	2	3	
	0.80	2	3	
	0.75	2	3	
2	0.95	1	2	2
	0.90	1	2	
	0.85	1	2	
	0.80	1	2	
	0.75	1	2	

Table 8.2 Initial hidden layer size estimation procedure applied to the parity problem (parity order 2 to 4, hidden layer size comparison)

Parity Order	angular distance threshold r_t	No. of Epochs (Enhanced algorithm)	No. of Epochs (Original Algorithm)
7	0.95	777	405
	0.90	739	
	0.85	447	
	0.80	414	
	0.75	425	
6	0.95	629	585
	0.90	585	
	0.85	211	
	0.80	306	
	0.75	247	
5	0.95	245	457
	0.90	135	
	0.85	135	
	0.80	98	
	0.75	264	

Table 8.3 Initial hidden layer size estimation procedure applied to the parity problem (parity order 5 to 7, training speed comparison)

Parity order	angular distance threshold r_t	No. of Epochs (Enhanced algorithm)	No. of Epochs (Original algorithm)
4	0.95	147	296
	0.90	147	
	0.85	147	
	0.80	147	
	0.75	147	
3	0.95	153	228
	0.90	153	
	0.85	153	
	0.80	153	
	0.75	153	
2	0.95	23	21
	0.90	23	
	0.85	23	
	0.80	23	
	0.75	23	

Table 8.4 Initial hidden layer size estimation procedure applied to the parity problem (parity order 2 to 4, training speed comparison)

In our simulation studies, the values of r_t from 0.95 to 0.75 are used. In general , a lower value of r_t will correspond to a larger initial hidden layer size as can be seen in the simulation results, since more neural basis are included in the initial neural basis set **N**. It is seen that the current estimation process ,when applied to the parity problem, tends to underestimate the initial network size required to solve the problem. thus for most of the cases additional nodes are required for the network to converge. However, since the initial hidden layer in general contains more than one hidden node especially for the higher-order parity problems, the convergence speed is much higher than the case when the node addition algorithm starts with a single node network, as can be seen from the results. We can in particular notice that when r_t=0.8, the initial hidden layer size estimation for the higher-order parity is relatively close to the optimum network size for solving that problem, and the final network corresponding to this angular distance threshold does not constitute much of an overestimation of the hidden layer size. Therefore, we can determine empirically that r_t=0.8 constitutes a suitable value for this initial hidden layer size estimation exercise. For large r_t, the situation is almost the same as for the unenhanced scheme and no improvement is apparent. For small r_t, the initial hidden layer size is usually large and the final network tends to be an overestimation of the actual network required to solve the problem. For low-order parity problems, the estimated initial hidden layer size is small due to the small number of training patterns which in turn implies a small number of neural basis in the initial neural basis set **N**

(2) Handwritten Character Recognition Set

The estimation scheme, when applied to the handwritten character recognition problem, gives the following results. In both the present problem and the time series modelling problem, we have utilized the generalization measure monitoring scheme with the respective generalization coefficient which is optimum for each problem as shown in Chapter 7.

angular distance threshold r_t	No. of epochs	Initial hidden layer size	Final hidden layer size	Recognition rate (training set)	Recognition rate (test set)
0.95	290	7	18	95.6%	84.4%
0.90	397	10	20	97.2%	80.0%
0.85	197	13	20	96.7%	82.2%
0.80	140	17	17	97.2%	87.2%
0.75	93	21	21	93.8%	85.6%

Table 8.5 The initial hidden layer size estimation procedure as applied to the handwritten character recognition problem

Error Gradient Threshold	No. of Epochs	No. of Nodes	Recognition rate (training set)	Recognition rate (test set)
0.05%	1189	30	97.8%	78.3%
0.06%	1365	26	97.2%	78.9%
0.07%	960	27	96.1%	80.0%
0.08%	850	26	96.1%	81.7%
0.09%	818	27	97.2%	78.9%
0.10%	656	27	96.1%	78.9%

Table 8.6 The original deterministic algorithm as applied to the handwritten character recognition problem

From the results it is seen that the hidden layer size increases with the angular distance threshold as expected. Moreover, it is found that the estimation for the initial hidden layer size corresponds roughly to the hidden layer size estimated by the unenhanced algorithm which starts with a single hidden node, thus establishing the validity of the range of r_t chosen. Moreover, it is noticed that the convergence speed is much greater than that for the unenhanced node addition scheme which starts with a single hidden node, and almost matches the convergence speed of BP for this particular problem, thus indicating the effectiveness of this estimation scheme. It is also noticed that the generalization capability of the network has increased when compared with the unenhanced scheme since the early consolidation of the knowledge of the training set in a small network is avoided. For values for r_t between 0.85 and 0.95, additional hidden nodes are required to complete the training task, while for r_t between 0.8 and 0.75, no additional hidden nodes are required. We again see that for this problem, the parameter r_t=0.8 constitutes a suitable value for the initial hidden layer estimation process, since it corresponds to the smallest initial network where no further node additions are required. Moreover, this value of r_t corresponds to the highest recognition rate on the test set.

(3)Time Series Modelling

The present estimation scheme, when applied to the time series modelling problem, gives the following results:

angular distance threshold r_t	No. of Epochs	Initial hidden layer size	Final hidden layer size	NRMSE
0.95	1517	4	10	0.1086
0.90	781	12	12	0.1033
0.85	549	14	16	0.1005
0.80	741	17	19	0.1037
0.75	1585	20	21	0.1018

Table 8.7 The initial hidden layer size estimation procedure applied to the time series modelling problem

Error Gradient Threshold	No. of Epochs	No. of nodes	NRMSE
0.05%	1520	4	0.1033
0.06%	1457	4	0.1032
0.07%	1413	4	0.1031
0.08%	1365	5	0.1041
0.09%	1337	5	0.1052
0.10%	1276	6	0.1090

Table 8.8 The original deterministic algorithm as applied to the time series modelling problem

From the results, it is noticed that for most values of r_t , the initial network constitutes an overestimation of the number of hidden nodes required to solve the problem except for r_t=0.95, where the correct hidden layer size of 4 as estimated by the unenhanced algorithm is generated. In general, further node additions are required to generate the final network, indicating that the current training set constitutes a complex problem even for an initial multi-node network. Convergence speed reduction is achieved when r_t is within the range of [0.8,0.9] at the expense of a large hidden layer size. It is interesting to note that an overly large network would, on the contrary, decrease the convergence speed as for the cases when r_t=0.8 and r_t=0.75. As a result, the most suitable value of r_t for the above problem has shifted from 0.8 from the previous two problems to 0.85 in terms of the achievement of the greatest convergence speed. In the case for r_t=0.95, though the correct minimum size of the network for solving this problem is estimated, the eventual network size generated by the dynamic node creation algorithm is still large. As a result, the merit of the current estimation procedure when applied to the time series modelling problem lies in the speeding up of the training process rather than the correct estimation of the initial hidden layer size.

To summarize, the original purpose of the initial hidden layer size estimation procedure is achieved through the parity problem and the handwritten character recognition problem in that an approximately correct initial hidden layer is estimated and a real increase in convergence speed is achieved. For the time series modelling problem, the original purpose is only partly achieved since only the speed up portion is achieved: the estimation procedure in general produces an overestimation for the hidden layer size. While the time series modelling problem belongs to the category of function approximation while the previous two problems belong to the category of classification problems, the above problem for the time series modelling training set may be due to the reason that we are using the neural basis, which is binary in nature, as a model for the hidden T-vectors. As a result, this model may be more suitable in representing binary training set than for real-valued training set. We have adopted this approach in estimating the initial hidden layer size due to its simplicity. It is hoped that further research would reveal a more generalized model which caters equally well for real-valued training set.

8.5 Concluding Remarks

In this Chapter we have derived a method in estimating the initial hidden layer size of the network prior to the application of the dynamic node creation procedure. The motivation for the derivation of this procedure lies in the fact that the starting network for most dynamic node creation algorithm which consists of a single hidden node is often unable to cater for most real world training sets. As a result, it would be useful if we can obtain an initial estimate of the hidden layer size using the information from the training set. This procedure is particularly relevant for those training sets requiring a relatively large network, in which case a great many number of node addition procedures can be saved. This procedure is applied to the three training sets presented in the previous chapters with various angular distance threshold T. In general, if a fixed threshold T is applied to all the training sets, there would be cases of underestimation of the initial hidden layer size in some training cases, while for the same threshold there would be cases of overestimation for other training sets. Therefore, the question of how to select the optimum T is still unsolved. However, for angular distance threshold T within the range of [0.75, 0.95], the resulting initial hidden layer size of the network all lies within reasonable limits and is consistent with the hidden layer size generated by the deterministic dynamic node creation algorithm from a single-node network. As a result, the present estimation approach would serve as a promising enhancement to all dynamic node creation algorithms.

9 CONCLUSION

9.1 Contributions

We would now summarize the contributions resulting from our course of research on dynamic node creation algorithms:

1. Formalization of the T-vector approach: The use of T-vectors as an alternative approach of summarizing the parameters of a neural network has been applied by a number of researchers [2,7,14] as an alternative to the traditional spatial vector or S-vector approach. This approach has the advantage that the representation capability of a neural network can be visualized in the form of the relationship between a few T-vectors and their respective linear spaces, instead of visualizing these relationships in the S-vector domain where the number of entities to be visualized is numerous. However, the past researchers applied this approach in an ad hoc manner to solve their own particular problems. The current research work attempts to place this alternative representation scheme in a formal framework. The name temporal vector or T-vector is first assigned to this alternative arrangement scheme to characterize this particular approach and to distinguish this approach from the usual parameter characterization method, which we have named the spatial vector or S-vector approach. The inverse desired output space $\mathbf{D}^{-1}$ and its spanning inverse desired output T-vectors $\mathbf{d}_k^{-1}$, the hidden space $\mathbf{H}$ and its spanning hidden T-vectors $\mathbf{h}_j$, the input space $\mathbf{X}$ and its spanning input T-vectors $\mathbf{x}_i$, are introduced to summarize the activities of the network nodes at the various layers of the network. The result of this formalization provides new insights in the relationship between the hidden space $\mathbf{H}$ and the input space $\mathbf{X}$, which other researchers have not explored before. Moreover, this formalization results in the most general formulation of the convergence criterion of a neural network in terms of the linear independency requirement for the hidden T-vectors and the subsequent suitable assignment of the output weights of the network. This new insight is instrumental in the formulation of the deterministic dynamic node creation algorithm.

2 The Formulation of a general node initialization scheme under the dynamic node creation environment: the dynamic node creation approach has been adopted to counteract the problem of indeterminate architecture. However, the various dynamic node creation algorithm proposed are somewhat restricted either in the scope of problems that it can cater for or in its own convergence capability. For example, some algorithms only cater for binary-valued training set [13,34], some algorithms do not possess a convergence proof [12], and some algorithms cannot be conveniently applied under an incremental learning environment [9]. In view of these, the growth algorithm and the deterministic dynamic node creation algorithms proposed in this thesis can be regarded as more general dynamic node creation algorithms when compared with the above algorithms in the sense that the two new algorithms are both able to fill in the missing capabilities of the above algorithms. In addition, the initialization of the parameters of the new hidden node in the deterministic algorithm depends on multiple training patterns which is thus a more preferable scheme than the growth algorithm in which the new hidden node parameters depend on a single training pattern, in that the memorization of a single noisy training pattern by the new hidden node is discouraged. In this way, the deterministic dynamic node creation algorithm can serve as the prototype for future models of similar kinds of algorithms.

3. Enhancement to the Dynamic Node Addition Process: The dynamic node addition process, which can partially alleviate the problem of indeterminate architecture, is unable to replace the conventional BP training process as the dynamic node addition process has bring some additional problems of its own. For example, the node addition scheme would require the monitoring of the error gradient parameter $\Delta E/E$. The termination of the BP fine-tuning process is signified when this parameter falls below a certain threshold. This choice of this threshold would in general affect the final size of the network generated: a low threshold would result in a smaller final network, while a larger threshold would result in a larger final network. It would seem that small gradient threshold would lead to substantial saving of resources. However, the corresponding prolonged training would lead to a deterioration in the generalization capability of the network. In the current research work, an attempt is made to monitor the generalization measure derived by Drucker et al [11] and detect its minimum values during the course of BP fine-tuning instead of the usual monitoring of the error gradient such as to prevent the overfitting of the training data due to prolonged training. In addition, the traditional dynamic node creation algorithms usually start with a single-node network. If a complex training set is encountered, it would in general take many steps of node addition before the network size is large enough to cater for the training set, which translates into a much prolonged training time compared with the same network being trained with the conventional BP approach. As a result, it would be beneficial if we can give a rough estimate of the initial hidden layer size of the network required to represent a particular training set, such that the final network would be built up using the smallest number of node additions. The last stage of our course of research attempts to derive such an initial hidden layer size estimation scheme such that a single procedure can be applied to a wide range of training data set in estimating the initial hidden layer size instead of using ad-hoc measures to derive this estimate for each individual training set. These two enhancements of the dynamic node creation algorithms would pave the way for the complete replacement of the conventional training approach with this new node creation approach.

9.2 Suggestions for Further Research

(1) A "Parameter-Free" training Algorithm: Besides the usual variable parameters of gain and momentum in the BP fine-tuning process, the deterministic dynamic node creation algorithm has introduced several new parameters. For example, the error gradient threshold is such a parameter which may affect the eventual size of the network. In general, such additional parameters are not desirable as they have to be supplied externally and the relationship between these parameters and the training performance of the network is often difficult to determine. In other words, the full automation of the training process is not possible with the existence of these parameters, and human intervention must be involved in controlling the training process. It would be ideal if we can extract all the relevant training information from the training set itself without any other independent information. The monitoring of the generalization measure during the BP fine-tuning process instead of the error gradient is the first attempt in removing the need for determining the error gradient threshold, and subsequent research efforts are required to remove the other training parameters.

(2) A more exact initial hidden layer size estimation: On realizing the various problems associated with starting the dynamic node creation training process with a single node network, we have derived an initial hidden layer size estimation procedure such that in

general fewer node addition are required to achieve the same approximation error as for the case when the initial network consists of only one hidden node. However, it is observed that this estimation procedure still depends on an angular distance threshold r_t which affects the initial size of the network. The utilization of this approach represents a great simplification in the estimation of the hidden layer size as we now only require to search through a relatively narrow range of the angular distance threshold for every training set instead of estimating in an arbitrary way the hidden layer size for each individual training set. However, there is still a degree of indeterminacy in the estimation procedure due to the finite possible range of the angular distance threshold. Experimentally, it was discovered that a threshold value of 0.8 would be applicable to a wide range of training sets and the resulting initial network size estimation would fall within a reasonable limit. However, this threshold value may not be suitable for some other training sets as we have only explored a limited number of training sets in our simulation studies. In general, it would be ideal if we can determine this angular distance threshold solely from the training set itself without resorting to any external determination of this parameter. In addition, we have made the approximation that each hidden node approximately emits one of the neural basis $\mathbf{n}_j$ in order that the initial hidden layer size estimation can be carried out. There would be the need for further explorations into the characteristics of the hidden nodes of a neural network in order to remove the above restriction such that a more accurate initial network state can be estimated which in turn leads to a shorter convergence time.

(3) The search for the fully deterministic training algorithm: the conventional BP approach of training the neural network does not possess any deterministic elements, the initial state of the network is generated wholly in a random fashion and the final state of the network is determined wholly by an iterative gradient descent procedure. As a result, we do not in general know the initial position of the current state of the network on the error surface and thus the random initialization does not constitute an effective search over the weight space for the global minimum. The current deterministic dynamic node creation algorithm attempts to assign initial states for every new hidden node added to the network based on information from the training set such that the eventual convergence of the network will be achieved. This deterministic assignment procedure can be interpreted as the placement of the current state of the network in the vicinity of the global minimum. However, due to the various

approximations made during the deterministic assignment procedure, we have to follow this procedure with a BP fine-tuning process in order to obtain the exact parameters for the network. Moreover, this approximation is restricted to those networks possessing a monotonic increasing nonlinear function and would not be appropriate for some other kinds of nonlinearities. A main reason for the necessity of this approximation is due to the presence of the hidden nonlinearities in the hidden node which complicates any attempts in analyzing the activities of the hidden node.

As a result, research efforts should be directed towards the detailed investigation of the hidden nonlinearity. This would at first seem to be a daunting task as we are crossing into the realm of nonlinear analysis. However, as the number of nonlinearities which are used in neural networks is rather restricted, we can attempt to concentrate on investigating the properties of these few kinds of nonlinearities which is a much more simplified task than attempting to analyze the general properties of nonlinearities. These investigations would help to provide better and better approximations to the real situation in the deterministic assignment procedure and would eventually lead to a truly deterministic algorithm which is free of the need of an iterative learning procedure.

REFERENCES

[1] S.I. Amari, "Mathematical Foundations of Neurocomputing," *Proc. IEEE*, vol.78, no.9, pp.1443-1463, September 1990.

[2] F. Barmann and F. Biegler-Konig ,"On a Class of Efficient Learning Algorithms for Neural Networks," *Neural Networks*, vol.5,pp.139-144,1992.

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

[4] S. Becker and Y. Le Cun, "Improving the Convergence of Back-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 Kaufmann, 1989.

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

[6] 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 Kaufmann, pp.519-526, 1989.

[7] S. Chen, C.F.N. Cowan, and P.M. Grant, "Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks," *IEEE Trans. on Neural Networks*, vol.2, no.2, pp.302-309, March 1991.

[8] J.Y. Choi and C.H. Choi, "Sensitivity Analysis of Multilayer Perceptron with Differentiable Activation Functions," *IEEE Trans. on Neural Networks*, vol.3, no.1, pp.101-107, January 1992.

[9] F.L. Chung, "Dynamic Construction of Back-Propagation Artificial Neural Networks," *MPhil Thesis*, The Chinese University of Hong Kong, 1991.

[10] W.J. Daunicht, "DEFAnet - A Deterministic Neural Network Concept for Function Approximation," *Neural Networks*, vol.4, pp.839-845, 1991.

[11] H. Drucker and Y. Le Cun, "Improving Generalization Performance Using Double Backpropagation," *IEEE Trans. on Neural Networks*, vol.3, no.6, pp.991-997, November 1992.

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

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

[14] O. Fujita, "A Method for Designing the Internal Representation of Neural Networks and Its Application to Network Synthesis," *Neural Networks*, vol,.4, pp.827-837, 1991.

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

[16] A.R. Gallant and H. White, "On Learning the Derivatives of an Unknown Mapping With Multilayer Feedforward Networks," *Neural Networks*, vol.5, pp.129-138, 1992.

[17] B. Giraud, L.C Liu, C. Bernard, and H. Axelrad, "Optimal Approximation of Square Integrable Functions by a Flexible One-Hidden-Layer Neural Network of Excitatory and Inhibitory Neuron Pairs," *Neural Networks*, nol.4, pp.803-815, 1991.

[18] M. Gori and A. Tesi, "On the Problem of Local Minima in Bachpropagation," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol.13, no.1, pp.76-86, January 1992.

[19] 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.

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

[21] 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.

[22] L. Holmstrom and P. Koistinen, "Using Additive Noise in Back-Propagation Training," *IEEE Trans. on Neural Networks*, vol.3, no.1, pp.24-37, January 1992.

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

[24] S.C. Huang and Y.F. Huang, "Bounds on the Number of Hidden Neurons in Multilayer Perceptrons,"*IEEE Trans. on Neural Network*,, vol.2 no.1, pp.47-55, January 1991.

[25] Y. Ito, "Approximation of Continuous Functions on R^d by Linear Combinations of Shifted Rotations of A Sigmoid Function With and Without Scaling," *Neural Networks*, vol.5, pp.105-115, 1992.

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

[27] W.H. Joerding and J.L. Meador, "Encoding A Priori Information in Feedforward Networks," *Neural Networks*, vol.4, pp.847-856, 1991.

[28] E.D. Karnin, "A Simple Procedure for Pruning Back-Propagation Trained Neural Networks", *IEEE Trans. on Neural Networks*, vol.1, pp.239-242, 1990.

[29] 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.

[30] 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.

[31] T. Lee and H.K. Kwan, "Experiments on Neural Net Recognition of Handwritten Character," *Proc. Inter. Multiconference Neural Networks, Modelling and Simulation, Knowledge Processing*, Chicago (USA), vol.1, pp.15-24, 1992

[32] M. Mackey and L. Glass, "Oscillation and Chaos in Physiological Control System," *Science*, pp.197-287, 1977.

[33] K. Matsuoka, "Noise Injection into Inputs in Back-Propagation Learning," *IEEE Trans. on Systems, Man, and Cybernetics*, vol.22, no.3, pp.436-440, May/June 1992.

[34] 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.

[35] D.F. Michaels, "Internal Organization of Classifier Networks Trained by Backpropagation,"*Int J. of Pattern Recognition and Artificial Intelligence*, vol 6, no.1, pp. 63-92, 1992.

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

[37] J. Moody and C. Darken, "Fast-learning in Networks of Locally-tuned Processing Units," *Neural Computation*, vol.1, no.2, pp. 281-294, 1989.

[38] M. Mougeot, R. Azencott and B. Angeniol ,"Image Compression With Back Propagation: Improvement of the Visual Restoration Using Different Cost Functions," *Neural Networks*, vol.4, pp. 467-476, 1991.

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

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

[41] B. Noble and J.W. Daniel, *Applied Linear Algebra*. New Jersey:Prentice-Hall, 1988.

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

[43] D.C. Park, M.A. El-Sharkawi, and R.J. Marks II ,"An Adaptively Trained Neural Network," *IEEE Trans. on Neural Networks*, vol. 2, no.3, pp,334-345, May 1991.

[44] J. Park and I.W. Sandberg, "Universal Approximation Using Radial-Basis-Function Networks," *Neural Computation*, vol.3, pp.246-257, 1991.

[45] M.M. Polycarpou and P.A. Ioannou, "Learning and Convergence Analysis of Neural-Type Structured Networks," *IEEE Trans. on Neural Networks*, vol.3, no.1, pp.39-50, January 1992.

[46] S.Z. Qin, H.T. Su, and T.J. McAvoy, "Comparison of Four Neural Net Learning Methods for Dynamic System Identification," *IEEE Trans. on Neural Networks*, vol.3, no.1, pp.122-129, January 1992.

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

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

[49] 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.

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

[51] R.S. Scalero and N. Tepedelenlioglu, "A Fast New Algorithm for Training Feedforward Neural Networks," *IEEE Trans. on Signal Processing*, vol,.40, no.1, pp.202-210, January 1992.

[52] J. Sietsma and R.J.F. Dow, "Creating Artificial Neural Networks That Generalize," *Neural Networks*, vol.4, pp.67-79, 1991.

[53] M.F. Tenorio and W. T. Lee, "Self-Organizing Network for Optimum Supervised Learning," *IEEE Trans. on Neural Networks*, vol.1, no.1, pp.100-110, March 1990.

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

[55] A.C. Tsoi, "Multilayer Perceptron Trained Using Radial Basis Functions," *Electronic Letters*, vol.25, no.19, pp.1296-1297, September 1989.

[56] A. Waibel, T. Hanazawa, G.E. Hinton, D. 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.

[57] A.R. Webb and D. Lowe, "The Optimised Internal Representation of Multilayer Classifier Networks Performs Nonlinear Discriminant Analysis," *Neural Network*, vol.3, pp.367-375, 1990

[58] A.S. Weigend, B.A. Huberman, D.E. Rumelhart, "Predicting the Future: A Connectionist Approach," *Int. J. Neural Syst.*, vol.1, no.3, pp. 193-209, 1990.

[59] M.K. Weir, "A Method for Self-Determination of Adaptive Learning Rates in Back Propagation," *Neural Networks*, vol.4, pp.371-379, 1991.

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

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

APPENDIX

The derivation of the BP algorithm will now be given by evaluating the derivatives of E with respect to the various network weights.

In Eq (2.8)

$$\begin{aligned}\frac{\partial E}{\partial u_{jk}} &= \sum_{t=1}^{P} \frac{\partial E}{\partial b_k(t)} \frac{\partial b_k(t)}{\partial u_{jk}} \\ &= \sum_{t=1}^{P} \frac{\partial E}{\partial y_k(t)} \frac{\partial y_k(t)}{\partial b_k(t)} \frac{\partial b_k(t)}{\partial u_{jk}} \\ &= \sum_{t=1}^{P} (d_k(t) - y_k(t)) f'(b_k(t)) h_j(t)\end{aligned} \tag{A.1}$$

where the derivative $f'(b_k(t))$ is given by

$$f'(b_k(t)) = y_k(t)(1 - y_k(t)) \tag{A.2}$$

Defining

$$\delta_k^o(t) = y_k(t)(1 - y_k(t))(d_k(t) - y_k(t)) \tag{A.3}$$

we obtain the results of Eq (2.8)

In Eq (2.9)

$$
\begin{aligned}
\frac{\partial E}{\partial w_{ij}} &= \sum_{t=1}^{p} \frac{\partial E}{\partial h_j(t)} \frac{\partial h_j(t)}{\partial w_{ij}} \\
&= \sum_{t=1}^{p} \left(\sum_{k=1}^{m} \frac{\partial E}{\partial y_k(t)} \frac{\partial y_k(t)}{\partial h_j(t)} \right) \frac{\partial h_j(t)}{\partial w_{ij}} \\
&= \sum_{t=1}^{p} \left(\sum_{k=1}^{m} \frac{\partial E}{\partial y_k(t)} \frac{\partial y_k(t)}{\partial b_k(t)} \frac{\partial b_k(t)}{\partial h_j(t)} \right) \frac{\partial h_j(t)}{\partial w_{ij}} \\
&= \sum_{t=1}^{p} \left(\sum_{k=1}^{m} \delta_k^o(t) u_{jk} \right) \frac{\partial h_j(t)}{\partial a_j(t)} \frac{\partial a_j(t)}{\partial w_{ij}} \\
&= \sum_{t=1}^{p} f'(a_j(t)) x_i(t) \sum_{k=1}^{m} \delta_k^o(t) u_{jk} \\
&= \sum_{t=1}^{p} h_j(t)(1-h_j(t)) \sum_{k=1}^{m} \delta_k^o(t) u_{jk}
\end{aligned}
\tag{A.4}
$$

Defining

$$
\delta_j(t) = h_j(t)(1-h_j(t)) \sum_{k=1}^{m} u_{jk} \delta_k^o(t) \tag{A.5}
$$

We obtain the results of Eq (2.9)