Theory and applications of artificial neural networks

Chen, Jian-Rong

Theory and Applications of Artificial Neural Networks

Jian-Rong Chen
School of Engineering and Computer Science
University of Durham

September 1991

A thesis submitted for the degree of
Doctor of Philosophy of the University of Durham

Abstract

In this thesis some fundamental theoretical problems about artificial neural networks and their application in communication and control systems are discussed. We consider the convergence properties of the Back-Propagation algorithm which is widely used for training of artificial neural networks, and two stepsize variation techniques are proposed to accelerate convergence. Simulation results demonstrate significant improvement over conventional Back-Propagation algorithms. We also discuss the relationship between generalization performance of artificial neural networks and their structure and representation strategy. It is shown that the structure of the network which represent a priori knowledge of the environment has a strong influence on generalization performance. A Theorem about the number of hidden units and the capacity of self-association MLP (Multi-Layer Perceptron) type network is also given in the thesis. In the application part of the thesis, we discuss the feasibility of using artificial neural networks for nonlinear system identification. Some advantages and disadvantages of this approach are analyzed. The thesis continues with a study of artificial neural networks applied to communication channel equalization and the problem of call access control in broadband ATM (Asynchronous Transfer Mode) communication networks. A final chapter provides overall conclusions and suggestions for further work.

Acknowledgements

I would like to express my thanks to my supervisor at the University of Durham, Professor P. Mars for his insightful guidance, enthusiastic encouragement and extreme patience in dealing with almost endless spelling mistakes. I would also like to thank members of Professor Mars' research group in Durham University, especially Richard, Chris and Roy, for their generous help. This research was carried out under a Sino-British Friendship Schoolarship which is sponsored by the British Council and the State Education Commission of P.R.China, to whom I would expess my appreciation. Finally, I should thank Mrs. Sylvia Mellanby for her hospitality and patience.

Declaration

I hereby declare that the work reported in this thesis was undertaken by myself, that the research has not been previously submitted for a degree, and that all sources of information have been duly acknowledged.

Copyright

Table of Contents

Chapter One
Introduction

1.1 Why Artificial Neural Networks ?

In the past few decades, the development of conventional digital computers has achieved great success. The application of digital computers has extended into almost every aspect of our life. Digital computers are extremely good at numerical calculation, formal logical inference and storage of data in bulk. They are significantly inferior to the human brain in such tasks as vision, speech and information retrieval based on content. This big performance difference between digital computers and the human brain is partly due to their different organizational structure and computational process.

Compared with digital computers, the operation speed of the human brain is much slower. If the transmission of an impulse by a neuron is considered as a fundamental step of computation, then the operation speed of a neuron is in the range from one step for a few seconds to several hundred steps per second[1], while a modern digital computer can carry out billions of operations per second. Although the human brain is slow, it can accomplish many cognitive tasks like visual recognition and speech understanding in about half of a second. Thus in [2], Feldman and Ballard suggested that these complex computation problems are solved in less than a hundred time steps. The explanation of this extremely high efficiency of computation of the human brain is its highly parallel structure. In the human brain, the number of connection for each neuron is varied from a few hundred to several thousand. The computation is carried out simultaneously in hundreds of thousands of parallel channels. Thus the research on neural networks is also called the connectionist approach. On the other hand, conventional digital

computers are mainly serial machines. Although parallel structures have been given more and more attention recently, the parallelism achieved is still far short of that in the human brain. In the brain in addition to parallel processing, the storage of information is also distributed.

Clearly, the human brain works in a fundamentally different way from that of conventional digital computers. The difference can be characterized by the highly parallel distributed organization and slow operation speed of the human brain and the generally serial structure and extremely fast operation speed of digital computers. The parallel distributed organization of the human brain not only gives it high efficiency in cognition, but also helps it to achieve a certain degree of fault tolerance. Thus it is natural to expect that, by doing research on artificial neural networks (or connectionist models), we may obtain a better understanding of mechanisms of the human cognition process, and possiblly synthesize new classes of intelligent machine. Research in neuroscience has made significant progress in the past decades, and many plausible models about human neural systems have been proposed. Development of high speed parallel processing now permit the simulation of large artificial neural networks. All these facts have contributed to the rapid boom of research on artificial neural networks in recent years. Basically there are two approaches to neural network research. One is concentrated on neuronal modeling which stress the biological plausibility of artificial neural networks. Another application orientated approach is more concerned in exploring the artificial neural networks as a parallel computing architecture or an adaptive system for application. In this case the biological plausibility is not emphasized. In this thesis the second approach is mainly adopted. In the next two sections, some basic concepts of neuroscience and artificial neural networks are introduced.

1.2 Basic Concepts of Biological Neural Networks

The brain is an extremely complex system. One of the fundamental achievements of neurosience in the early part of this century is the recognition that the neuron is the basic building unit of the brain, and neurons are interconnected with a high degree of order and specificity [1]. The number of neurons in a human brain is estimated around 10^{11}, and they are organized into many specialized regions for different functions [3]. A diagram of a neuron is shown in Fig-1.1. Although it is simplified, it captures some of the most important features of neurons.

The cell body of a neuron is called the soma. The spine-like extensions of the cell body are dendrites. They usually branch repeatedly and form a bushy tree around the cell body and provide connections to receive incoming signals from other neurons. The axon extends away from the cell body to provide a pathway for outgoing signals. Signals are transfered from one neuron to another through a contact point called a synapse. Although the synaptic junctions can be formed between axon and axon, between dendrite and dendrite and between axon and cell body, the most common synaptic junction is between the axon of one neuron and the dendrite of another. There are two classes of synapses. The excitatory synapse tends to promote the activation of neurons, while the inhibitory synapse play an opposite role. When a neuron is activated or firing (this could be caused by an external stimulus), an impulse signal travels down along the axon, until it reaches a synapse. At this point some kind of chemical transmitter is released to promote or inhibite the firing of the receptor neuron.

Some research shows that the excitablity and inhibition of synapses can be enhanced by the activities of neurons, and this synaptic plasticity is believed by many researchers to be the neuronal mechanism of learning and memory function

of the brain [4] [5]. Artificial neural networks is a the natural extension of this synaptic plasticity learning theory.

1.3 Basic Concepts of Artificial Neural Networks

An artificial neural network (ANN) consists of neurons, a connection topology and a learning algorithm. The neurons are also called units or processing elements etc. A typical unit is depicted in Fig-1.2. The input signals $(x_1, \ldots, x_n)$ come from either the external enviroment or outputs of other units in the network. Associated with each input connection link of the unit is an adjustable value called a weight or connection strength. This is a direct immitation of the synaptic plasticity which is described in the previous section, so in some neuroscience and psychology literature it is also called long-term memory (LTM). Usually w_{ij} represents the weight of the connection from unit i to unit j. θ_j is the threshold of the unit. The operation of the unit can be described as

$$y_j = f(\sum_{i=1}^{n} x_i w_{ij} - \theta_j) \qquad (1-1)$$

The operation can be time continuous if the ANN is implemented by analog hardware. In the discrete time case it can work in either synchronous mode or asynchronous mode.

The function $f(\ )$ in (1-1) is called an activation function. Four most commonly used activation functions are linear, ramp, step and sigmoid functions. The linear function shown in Fig-1.3(a) can be defined by

$$f(x) = \alpha x \qquad (1-2)$$

The ramp function shown in Fig-1.3(b) is described by

$$f(x) = \begin{cases} S, & \text{if } x > S; \\ x, & \text{if } |x| \leq S; \\ -S, & \text{otherwise.} \end{cases} \qquad (1-3)$$

where S is the saturated output. The function in (1-3) can be adapted into other forms by change of slope and saturation point. The limiting form of a ramp function is a step function which is depicted in Fig-1.3(c). It can be defined as

$$f(x) = \begin{cases} S, & \text{if } x \geq 0; \\ -S, & \text{otherwise.} \end{cases} \qquad (1-4)$$

The sigmoid function displayed in Fig-1.3(d) can take many different analytical forms. The following equations (1-5) and (1-6) are just two examples.

$$f(x) = \frac{1 - e^{-x}}{1 + e^{-x}} m \qquad (1-5)$$

$$f(x) = m \times sign(x)(1 - e^{-|x|}) \qquad (1-6)$$

where m is the magnitude coefficient. In some situations, when only positive outputs are needed, binary 1 or 0 for example, these activation functions can be shifted upward above the x-axis.

The connection topology largely takes two forms. These are the feed-forward only architecture and the feed-back or recurrent architecture. Many feed-forward ANN are fully connected, with no fine hierarchical built-in structure. But the connection topology plays an important role in improving the generalization and learning performance of ANN.

There are two classes of learning strategies for ANN. They are supervised learning and unsupervised learning. The learning or training algorithms will be discussed in more detail in the following chapters.

Although the basic idea of artificial neural networks comes from the study of biological nervous systems, most ANN capture only some basic features of biological nervous systems. This simplification can be justified by the lack of powerful mathematical tools for dealing with large nonlinear systems, in addition the simplification

still captures some most important features of biological nervous systems which are assumed to be the underlying mechanism of learning.

1.4 Outline of the Thesis

In this thesis we discuss some theoretical problems associated with artificial neural networks and their application in control and communication engineering. Chapter 2 is a brief review of some of the most commonly used artificial neural networks. In chapter 3 the convergence performance of the Back-Propagation algorithm is discussed, and two stepsize variation techniques are proposed for acceleration of its convergence speed. Chapter 4 addresses the generalization and representation problem of artificial neural networks. Chapter 5 to chapter 7 consider some potential applications of artificial networks. In chapter 5 some aspects of using MLP networks for system identification are discussed. Chapter 6 explores the application of a self-organization network as a structure for communication channel equalization, and chapter 7 propose two learning networks for the ATM call access control. Finally, in chapter 8 some overall concluding remarks and suggestions for further work are given.

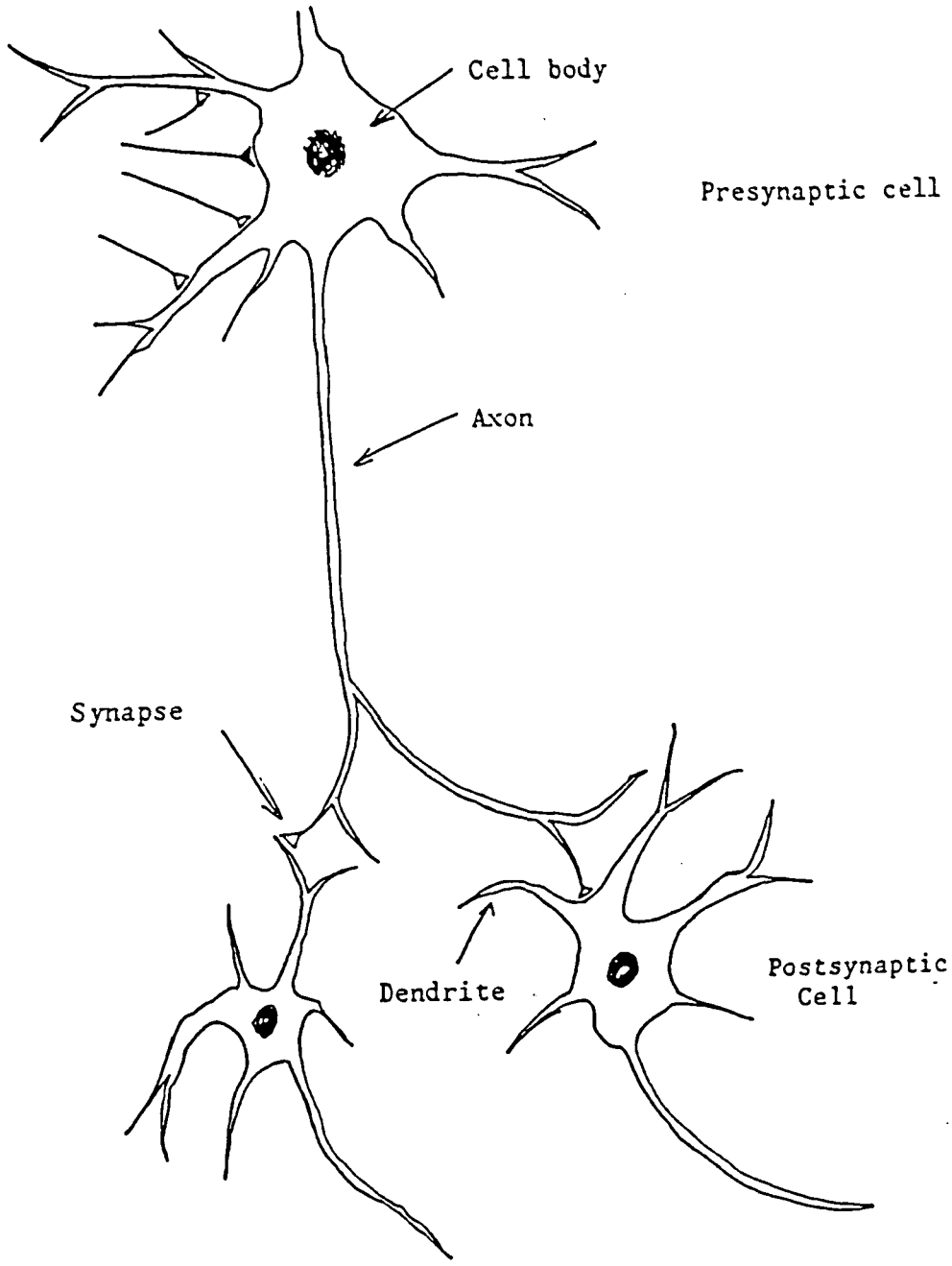


Fig-1.1 A simplified diagram of neurons

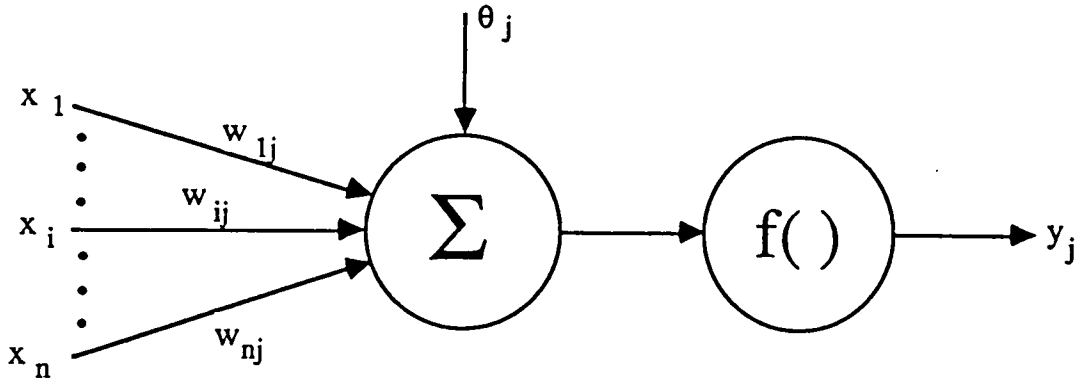


Fig-1.2 A general ANN unit

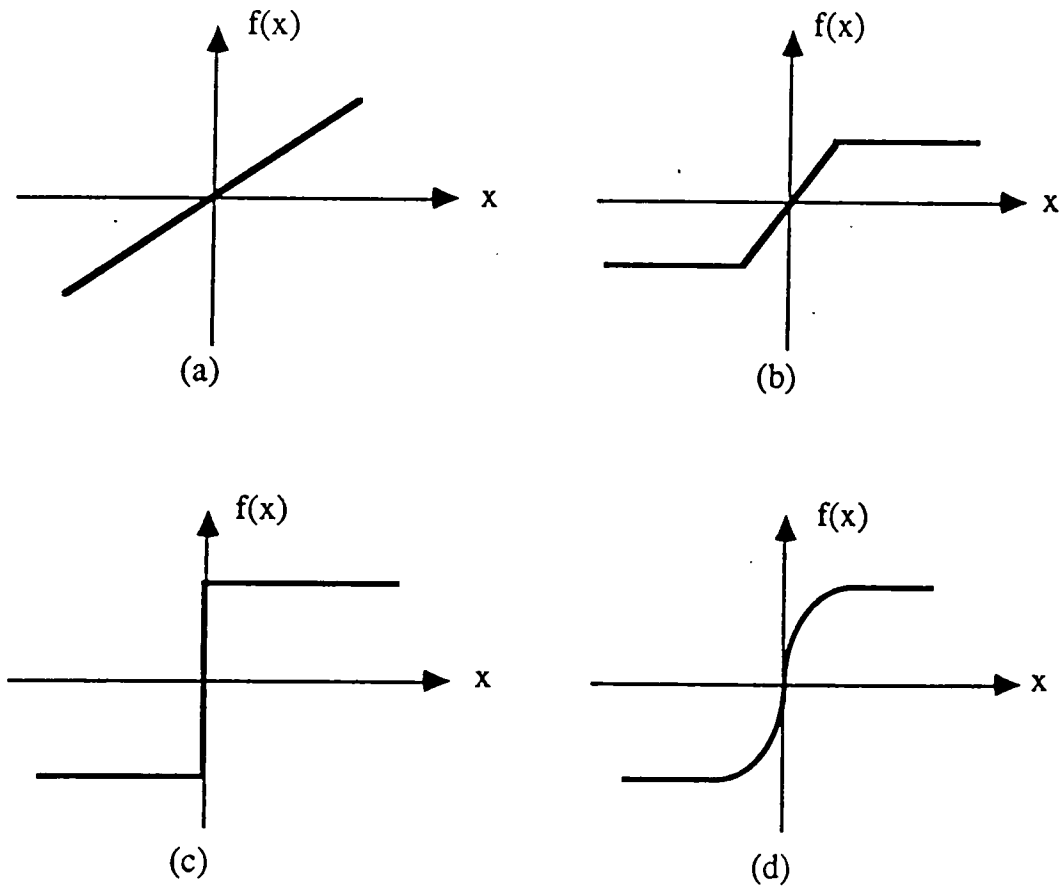


Fig-1.3 Activation functions of ANN

Chapter Two

A Review of Artificial Neural Networks

There are numerous artificial neural networks (ANN) that have been proposed in the research literature [6] [7] [8] [9]. In this brief review, only some widely known ANN are discussed. Based mainly on their learning schemes, the ANN can be classified into three divisions : (a) Supervised Learning ANN, (b) Unsupervised Learning ANN, and (c) Prewired or Hardwired ANN.

2.1 Supervised Learning ANN

Supervised learning literally means learning under external instruction. In the context of machine learning theory (include ANN), it means during the learning process, when the machine produces an output or action under specific stimulus, the full or partial information of the desirable output or action is available from the environment for learning or error correction (it is usually called weight updating in ANN learning). We now consider some of the most popular ANN learning algorithms which use supervised learning schemes.

2.1.1 Back-Propagation Algorithm

The Back-Propagation Algorithm is basically a gradient-descent algorithm which is widely used in numerical calculation and control and system engineering. The idea of using a gradient-descent algorithm as a learning algorithm can be traced back to the early work of Widrow [10] and Rosenblatt [11]. Their pioneering work was mainly concerned with single layer networks. The basic structure of these single layer networks is shown in Fig-2.1. It can be seen that it is a kind of adaptive linear combiner, its output is a linear combination of its input. Although the single

layer network has a very simple structure, it has achieved great success and wide acceptance in adaptive signal processing and control [12] [13] [14] [15]. However when it is used to model cognitive processes of the brain, it has been shown by Minsky and Papert that the single layer structure has some fundamental limitations [16].

The publication by Minsky and Papert significantly reduced the interest in ANN, although some researchers were still pursuing the field. In 1974 Werbos introduced the gradient-decent algorithm into the multi-layer networks [17] which are now usually called Multi-Layer Perceptrons (MLP). In particular, when Rumelhart, Hinton and Williams independently discovered the Back-Propagation Algorithm of MLP [18] and exploited the power and potential in great detail in [19], the research interest in this field has been revived. There have been numerous papers published on MLP and other ANN. However some fundamental problems associated with MLP still need to be investigated, and many potential applications need to be explored. In the following chapters, some properties of MLP and its applications will be considered.

While the majority of research activities are concentrated on using Back-Propagation Algorithm for training of feed-forward networks, like MLP, there is also some research which has been reported on recurrent or feedback networks [20] [21] [22] [23].

One conspicous way of dealing with input signals which have temporal structures is to represent time explicitly by associating the serial order of the input signals with the dimensionality of the input vector of a feed-forward network. The first temporal event is represented by the first element of the input vector, the second temporal event is represented by the second element, and so on. However

this scheme of dealing with temporal structure has inherent weakness. It is very similar to the FIR filter in digital signal processing, which can only model finite impulse response systems. Thus it lacks the ability of modeling the kinds of dynamic properties like self-sustained oscillation for example. Thus recurrent networks are indispensable for modeling dynamic behaviour. In addition, it is more biologically plausible, as recurrent paths occur frequently in real nervous systems [24].

In the training of recurrent ANN, the desirable output usually takes the form of trajectories in state space [22] [23] or a string of symbols [21]. So the objective function to be minimized takes the form

$$J(t_0, t_s) = \sum_{\tau=t_0}^{t_s} (\mathbf{D}(\tau) - \mathbf{O}(\tau))^T (\mathbf{D}(\tau) - \mathbf{O}(\tau)) \qquad (2-1)$$

where $\mathbf{D}(\tau)$ is the desirable output vector at time τ, and $\mathbf{O}(\tau)$ is the actual output vector of the recurrent network at time τ. It is very similar to the optimal trajectory problem in dynamic programming [25]. Actually, the dynamic programming technique can be used for training of recurrent networks [23]. For the $J(t_0, t_s)$ defined in (2-1), a gradient-descent based algorithm which is very similar to the Back-Propagation Algorithm for the feed-forward networks can be used for minimization Note that

$$\begin{aligned} J(t_0, t_s) &= (\mathbf{D}(t_0) - \mathbf{O}(t_0))^T (\mathbf{D}(t_0) - \mathbf{O}(t_0)) \\ &\quad + J(t_0 + 1, t_s) \end{aligned} \qquad (2-2)$$

let

$$e(\tau) = (\mathbf{D}(\tau) - \mathbf{O}(\tau))^T (\mathbf{D}(\tau) - \mathbf{O}(\tau))$$

then

$$J(t_0, t_s) = e(t_0) + J(t_0 + 1, t_s) \qquad (2-3)$$

So the gradient to weight vector $\mathbf{W}$ can be calculated as

$$\nabla_{\mathbf{W}} J(t_0, t_s) = \nabla_{\mathbf{W}} e(t_0) + \nabla_{\mathbf{W}} J(t_0 + 1, t_s) \qquad (2-4)$$

The value of $\nabla_{\mathbf{W}} e(\tau)$ is the weighted sum of all the previous gradients until time τ. This is the difference between Back-Propagation in recurrent networks and that in MLP. So from (2-4), $\nabla_{\mathbf{W}} J(t-0, t_s)$ can be calculated by summation of $\nabla_{\mathbf{W}} e(\tau)$.

In a strict sense, to minimize $J(t_0, t_s)$ in (2-1) by a gradient-decent algorithm, the weights can only be updated after the network goes through the whole training sequence $\mathbf{D}(\tau)(\tau = t_0, \ldots, t_s)$. However in practice, weights are usually updated at each time step. As long as the changing of weights is much slower compared with the time scale of dynamics of the network, it would not be a serious problem [23].

The main objective of exploring recurrent ANN is to make ANN context sensitive. In other words, its action should depend not only on present input stimulus, but also the previous context environment. This would require ANN to be able to generalize from its learning samples. As the general understanding of complex dynamic systems is still poor, this is a very difficult task. Compared with the progress in research on feed-forward ANN, there is still a great deal to be done for recurrent ANN.

In the case of multi-modal objective functions, the gradient-descent based algorithm has its local minima problem. One way of escaping from local minima is to use a stochastic technique. In the following two sections, two stochastic learning ANN will be discussed.

2.1.2 Boltzmann Machines

The idea of Boltzmann machines was introduced by Hinton, Ackley and Sejnowski [26] [27]. It is a kind of ANN which tries to satisfy a large number of constraints in a probabilistic sense. This is also called weak constraints satisfac-

tion.

The key point of the Boltzmann machine is to use a simulated annealing technique to estimate the state probability distribution functions either under environment constraints or in unconstrained conditions. Then the gradient-descent algorithm is used to reduce the discrepancy between these two distribution functions [28].

The idea of simulated annealing comes from physics [29] [30]. Annealing is the physical process used for the making of crystal. During the annealing process, the solid is first melted under very high temperature, then the temperature is decreased slowly, especially in the vicinity of the freezing point. The perfect crystal, or in another words a system with the lowest energy, can be made from this process. From a mathematical point of view it is a minimization process. The physical mechanism of the annealing process can be explained with the help of statistic mechanics.

According to statistic mechanics, for a substance which consists of a large number of atoms, when it reaches the thermal equilibrium at the temperature T, its state distribution can be described by a Boltzmann distribution. The probability of the substance to be in state i with energy E_i at temperature T is given by

$$P_T(x = i) = \frac{1}{s(T)} exp(\frac{-E_i}{k_B T}) \qquad (2-5)$$

where k_B is Boltzmanns constant, x is a random variable denoting the current state of the substance, $s(T)$ is defined by

$$s(T) = \sum_i exp(\frac{-E_i}{k_B T}) \qquad (2-6)$$

where the summation goes through all possible states. The annealing process can be simulated by a computer program [31]. In a computer program, the state transition

can be modeled by the following algorithm. At state i with energy E_i, the subsequent state j is generated by applying a perturbation mechanism which transforms the current state into the next state by a small distortion. If the energy difference $E_i - E_j$ is less than or equal to 0, the state j is accepted as the current state. If the energy difference is greater than 0, the state j is accepted with a certain probability which is given by

$$exp(\frac{E_i - E_j}{k_B T}) \qquad (2-7)$$

where T is the current temperature. If a lot of state transitions are simulated at a specific temperature T, then the thermal equilibrium is assumed to be reached at T.

From equation (2-5), it can be seen that when the substance is in a thermal equilibrium condition and the temperature is approaching zero, the probability of the substance remaining in the lowest energy state would be much greater than that for any other energy states. This is the physical mechanism of an annealing process.

The foregoing annealing mechanism can also be explored for optimization as the annealing is basically a mininization process. The Simulated annealing was first introduced into optimization by Kirkpatrick et. [29]. In simulated annealing the objective function has replaced the energy function E_i and E_j in (2-7). By properly reducing the emulated temperature, the minimum point of an objective function can be reached with probability 1 [28].

In a Boltzmann machine, simulated annealing is used for estimation of probability distribution functions. Fig-2.2 shows an example of a Boltzmann machine. It is an ANN. Some of its units which have direct connection with the environment are called environment contact units. Each unit in the network is a neuron like

processing element which can be described by

$$y_j = \begin{cases} 1, & \text{if } \sum_i x_i w_{ij} + \theta_j > 0; \\ 0, & \text{otherwise.} \end{cases} \qquad (2-8)$$

where x_i is input to the unit, w_{ij} is the weight of connection from unit i to unit j and θ_j is the threshold and y_j is the output. So it is a binary type ANN. Its objective function or energy function is defined by

$$E = -\sum_{i,j} w_{ij} y_i y_j + \sum_j \theta_j y_j \qquad (2-9)$$

As θ_j in (2-8) can be regarded as a connection which is connected to a permanently active unit, if this unit is assumed to be part of the network, then (2-9) can be rewritten as

$$E = -\sum_{i,j} w_{ij} y_i y_j \qquad (2-10)$$

Let $p(i)$ represent the probability of the machine in state i when its input and output units are clamped to training sample vectors, and it is in the thermal equilibrium under simulated annealing using energy function defined in (2-10). $p'(i)$ is the corresponding probability when the machine is running freely with no environment constraints. An information theoretic measure of the discrepancy between these two probability distributions can be given as

$$D = \sum_i p(i) ln \frac{p(i)}{p'(i)} \qquad (2-11)$$

D is zero if and only if two distributions are identical, otherwise it is positive [28]. The partial derivative of D to w_{ij} is given by [27]

$$\frac{\partial D}{\partial w_{ij}} = -\frac{1}{T}(p_{ij} - p'_{ij}) \qquad (2-12)$$

where p_{ij} is the probability of unit i and unit j both being in '1' state when the machine is running under environment constraints, that is to say it is clamped to

training samples. In equation (2-12) p'_{ij} is the corresponding probability when the machine is running freely with no environment constraints. The values of p_{ij} and p'_{ij} can be estimated by the foregoing simulated annealing process. Then the weights of the machine can be updated by the formula

$$\Delta w_{ij} = \alpha(p_{ij} - p'_{ij}) \qquad (2-13)$$

where α is the stepsize.

So the Boltzmann machine learning can be summarized as follows:

1 *The environment contact units of the machine are first clamped to learning samples under simulated annealing, and p_{ij} are estimated during the process. Then p'_{ij} are estimated under a free run condition when the environment contact units are not clamped.*

2 *Weights are updated using formula (2-13), then go back to the previous step until p_{ij} and p'_{ij} are close enough.*

The most serious drawback of Boltzmann machines is their extremely slow learning speed. Simulated annealing is a very slow process, and in Boltzmann machines, it has to be repeated many times. This makes it extremely slow. There has been some research on accelerating the learning speed of Boltzmann machines by using a Cauchy distribution [32] or using high-order energy functions [33]. The learning scheme using a Cauchy distribution is called a Cauchy machine, in which the Cauchy distribution replaces the Boltzmann distribution in (2-7). However all these learning schemes are based on the Monte Carlo method and are thus always slow compared to a determinstic approach.

2.1.3 Reinforcement Learning

In both Back-Propagation and Boltzmann machine learning, full information on the desirable output is required. However under some circumstances the environment only provides partial information of the desirable output, like a critic signal with reward/penalty responses to an output of a ANN. In these cases reinforcement learning has to be explored [34].

Reinforcement learning ANN is closely related to learning automata theory [35]. A learning automaton is an automaton that improves its performance while operating in a random environment. A simple binary environment can be defined as a triple $\{\alpha, c, \beta\}$ where $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ represents a finite input set, $\beta = \{0, 1\}$ represents a binary output set, '0' is an favourable or reward response, and '1' is an unfavourable or penalty response. $c = \{c_1, c_2, \ldots, c_n\}$ is a penalty probability set, which can be defined as

$$c_i = Pr(\beta(t) = 1 \mid \alpha(t) = \alpha_i) \quad (i = 1, 2, \ldots, n)$$

where t represents at discrete time t.

A simple variable structure stochastic automaton can be represented as a triple $\{\alpha, \beta, T\}$, where α and β are as defined for the environment, but for the automation α is an output set and β is an input set. A reinforcement learning automaton can be described as

$$p(t+1) = T[p(t), \alpha(t), \beta(t)] \qquad (2-14)$$

and

$$p(t) = \{p_1(t), p_2(t), \ldots, p_n(t)\}$$

where $p_i(t)$ is the probability of selecting action α_i at time t. A linear reward-penalty (L_{R-P}) learning algorithm for the automaton and the environment can be

expressed as

if $\alpha(t) = \alpha_i$ **and** $\beta(t) = 0$ $(reward)$

$p_i(t+1) = p_i(t) + a(1 - p_i(t));$

$p_j(t+1) = (1-a)p_j(t) \quad (j \neq i);$

else if $\beta(t) = 1$ $(penalty)$

$p_i(t+1) = (1-b)p_i(t);$

$p_j(t+1) = (1-b)p_j(t) + \frac{b}{n-1} \quad (j \neq i)$

The properties of L_{R-P} learning has been discussed in great detail in [35], and it has been applied to telecommunications [36] and decision making [37] and other applications.

The Associative Reward/Penalty (A_{R-P}) learning ANN is an extension of the foregoing learning automata. It was first introduced as a neuron-like computing element by Barto [38] [39]. An A_{R-P} learning element is depicted in Fig-2.3. Each input link has an associated weight w_j, and the link labelled r is a pathway of reinforcement signal. The output y of an A_{R-P} element is calculated by

$$y(t) = \begin{cases} +1, & \text{if } \mathbf{W}^T(t)\mathbf{X}(t) + \eta(t)\, 0; \\ -1, & \text{otherwise.} \end{cases} \qquad (2-15)$$

where $\mathbf{W}^T(t) = [w_1(t), w_2(t), \ldots, w_n(t)]$ is the weight vector, $\mathbf{X}^T(t) = [x_1(t), x_2(t), \ldots, x_n(t)]$ is the input vector at time t, and $\eta(t)$ is any random variable taken from a specific distribution. The learning algorithm or the updating of weights can be

described as

$$\mathbf{W}(t+1) - \mathbf{W}(t) = \begin{cases} \rho(t)[y(t) - E\{y(t) \mid \mathbf{W}(t), \mathbf{X}(t)\}]\mathbf{X}(t), & \text{if } r(t) = 0; \\ \lambda\rho(t)[-y(t) - E\{y(t) \mid \mathbf{W}(t), \mathbf{X}(t)\}]\mathbf{X}(t), & \text{if } r(t) = 1. \end{cases} \tag{2-16}$$

where $0 \leq \lambda \leq 1$ and $\rho(t) > 0$, $r(t) = 0$ means reward and $r(t) = 1$ means penalty.

The convergence of A_{R-P} learning can be proved under certain conditions [39], and simulation shows that A_{R-P} learning performs quite well in certain test problems apart from slow learning speed [40]. However there is a fundamental limitation of the learning ability of single layer A_{R-P} network which is stated as a condition of convergence in [39]. It demands that the set of input vectors be a linear independent set. As is known, for an n-dimensional space there can only be at most n vectors in a linear independent set. Thus this condition excludes many complex learning tasks like the XOR problem out of the coverage of the convergence theorem. Some simulation studies have been reported on using A_{R-P} learning networks with multi-layer structures to deal with this problem [40]. However a more thorough study is still needed to address problems like convergence and learning capacity for multi-layer A_{R-P} learning ANN.

2.2 Unsupervised Learning ANN

The ANN discussed in the previous sections are all trained under the supervision of a external intructor which can give some information about the desirable response of the network. In this section, a class of ANN which learn by endogenous mechanism are discussed.

2.2.1 Associative Memory

In conventional digital computers, the storage and retrieval of information is

organized by spatial location (or address) in the hardware. On the other hand for the biological neural systems, the storage and retrieval of information is based on the collective interaction of a great number of neurons, so it seems more natural to assume that the representations of concepts and other pieces of information are stored in a distributed fashion as collective states of the neural systems rather than in a specific spatial point in the systems.

According to this non-local memory theory, the pieces of information are represented by the pattern of activity of many interconnected units instead of one unit. Each unit acts as an element in the representation of many pieces of information. This distributed representation scheme is a more plausible model of human memory than digital computers, as it has captured some features of human memory. For example content addressable memory, in which the retrieval of information is based on content instead of its spatial location. When only a partial pattern is available, the whole pattern can be reconstructed. This is called associative memory.

Obviously ANN are attractive for the implementation of distributed memory or associative memory. In [41] Hopfield suggested a binary network which can perform associative memory. The architecture of the network is shown in Fig-2.4, which is a fully connected network. The activation function of units in the network is defined as

$$y_i = \begin{cases} 1, & \text{if } \sum_{j \neq i} y_j w_{ji} > 0; \\ 0, & \text{otherwise.} \end{cases} \qquad (2-17)$$

where y_i is the output of unit i, w_{ji} is the strength of the connection from unit j to unit i. Information is stored by the learning algorithm as

$$w_{ji} = \sum_s (2y_j^s - 1)(2y_i^s - 1) \qquad (2-18)$$

where y_i^s is the ith element of vector $\mathbf{Y}^s$ $(s = 1, 2, \ldots, n)$, and $\mathbf{Y}^s$ are the state vectors to be stored in the network. w_{ii} are assumed zero.

For the retrieval of information, when a distorted state vector is presented to the network, each unit will synchronously or asynchronously update their states with activation function given in (2-17), until the network settles into a stable state. The stable state vector represents the retrieved piece of information.

The stability of the network can be proved using a Liapunov function method. The Liapunov function for the network can be defined as [41]

$$E = -\frac{1}{2} \sum_{\substack{i,j \\ i \neq j}} w_{ij} y_i y_j \qquad (2-19)$$

From (2-18) it is obvious that $w_{ij} = w_{ji}$, and if we consider the network activation function in (2-17), it can be proved that E defined in (2-19) is a positive monotonically decreasing function in the recall process of the network. Thus the network is stable. The stability theorem can be extended to the situation where $w_{ii} \neq 0$ and $w_{ii} \geq 0$ [42].

From the foregoing discussion it can be seen that a memory item or vector is represented by a stable state or an attractor of the network. So the memory capacity of the network is dependent on the dynamic properties of the network. How many stable states a network has determines how many item or vectors can be stored. The simulation study reported in [41] concluded that if the network has N units, then about 0.15N vectors or items can be simultaneously remembered without any severe retrieval error. A theoretical analysis based on statistical theory [42] shows the capacity of the network can be expressed as

$$m = \frac{(1-2\rho)^2}{2} \frac{n}{\log n} \qquad (2-20)$$

where n is the number of units in the network. This capacity can be explained as follows. If the items or vectors to be stored are choosen randomly with probabilty

0.5 for each bit of their elements to be 1 or 0, and they are statistically indepedent, then these vectors can all be correctly recalled with high probability by any recall vector which lies within the distance of ρn from the correct memory vector if their number is less than or equal to the m given in (2-20).

The capacity of the network can be increased by using pseudo-inverse techniques [43] or by storing correlated vectors [44] [45]. However correlated vectors may cause cross-talk when noise is present in the recall signals. Thus just making each memory vector an attractor or a stable state is not enough to guarantee satisfactory recall performance, the attraction basin of each stable state should be large enough to ensure a certain degree of noise immunity. The research reported in [44] also suggested that a larger attraction basin means smaller memory capacity. Thus it is a trade-off between the memory capacity and the noise immunity.

The associative memory ANN introduced by Hopfield is an autoassociative associative network. The bidirectional associative memory (BAM) suggested by Kosko [46] is a kind of heteroassociative associative memory network in which vector pairs $(\mathbf{A}_i, \mathbf{B}_i)$ are to be stored and recalled. When one element of the data pair is available, the other element can be recalled by running the network. In the following discussion, the dimensionality of $\mathbf{A}_i$ and $\mathbf{B}_i$ are assumed to be n and m respectively.

A schemetic diagram of a BAM network is presented in Fig-2.5. The connections in the network are all bidirectional. F_A is the layer which represents vector $\mathbf{A}_i$ and F_B is the layer for vector $\mathbf{B}_i$. If the weights of the connections are denoted by a matrix M which is $p \times n$, then when a vector A is presented to F_A, the recall

process is an iteration process described by

$$\begin{array}{ccccccc} \mathbf{A} & \longrightarrow & M & \longrightarrow & f & \longrightarrow & \mathbf{B} \\ \mathbf{A}' & \longleftarrow & f & \longleftarrow & M^T & \longleftarrow & \mathbf{B} \\ \mathbf{A}' & \longrightarrow & M & \longrightarrow & f & \longrightarrow & \mathbf{B}' \\ & & \cdots & & & & \end{array} \qquad (2-21)$$

where f is a nonlinear function or mapping. The iteration stops at a stable state. Hence stability of the system is a precondition for successful recall. It can be proved that if the threshold function (2-17) is used for as the nonlinear function f in (2-21), then the above iteration process is stable for any real matrix M. This strong stability property of a BAM network provides a great degree of flexibility for the design of an associative memory network. For the discrete BAM network introduced in [46], the weight matrix M can be obtained by

$$M = \sum_{i=1}^{N} \mathbf{B}_i \mathbf{A}_i^T \qquad (2-22)$$

or the weights can be calculated by an adaptive rule [47] as

$$\frac{dw_{ij}}{dt} = \alpha(-w_{ij} + S(a_i)S(b_j)) \qquad (2-23)$$

where $S(\ )$ is a sigmoid function.

From (2-21) it can be seen that the BAM network is a nonlinear dynamic system and analytical study of the network is non-trivial. Its storage properties are still under investigation [48] [49].

2.2.2 Adaptive Resonance Theory (ART)

One objective of the research on ANN is to develop the kind of intelligent machine which can categorize the input stimulus and self-organise according to its experience with an external environment. Competitive learning [50] is one approach to self-organization learning. A general competitive learning ANN can be described

by Fig-2.6, which has several layers. Within each layer, a winner-take-all strategy is used to cluster the input patterns. Features discovered in the first layer can be used as input patterns in the subsequent layers. However when a competitive learning network is exposed to a complex environment, it may suffer from a serious stability problem which is discussed by Grossberg [51]. The problem is that when the network goes through a series of learning samples, the learning may keep on going and never settle into a stable solution even when there are repeated patterns in learning samples. This is called the stability-plasticity dilemma. The purpose of the adaptive resonance theory (ART), which was first introduced by Grossberg [52] and elaborated by Carpenter and Grossberg [53], is to embed a self-regulating mechanism into competitive learning in such a manner as to provide stability.

An ANN based on Adaptive Resonance Theory is depicted in Fig-2.7. It is called an ART1 network [53]. The operation of ART1 can be described briefly as follows.

When an input pattern $\mathbf{A}_k = (a_1^k, a_2^k, \ldots, a_n^k)$ is presented to the bottom layer F_1, the units in F_1 will be activated and sustain an activation or short-term memory (STM). The activation in F_1 will be send through long-term memory (LTM, they are also called weights in other literature) connection w_{ij} to activate the top layer F_2. The units in F_2 will compete with each other until one unit wins the competition and inhibits the activation of all the other units. Then the winning unit sends back a top-down expectation pattern to F_1 through LTM connection v_{ij}. If the difference between the two patterns exceeds a vigilance parameter, then the classification would be regarded as incorrect, and F_2 layer would be switched to another unit, the previous winning unit would be disabled. If the difference is acceptable, then $\mathbf{A}_k$ would be classified to the class represented by the winning unit. The LTM is then updated.

The STM activity in F_1 can be described by a differential equation

$$\epsilon\frac{dx_i}{dt} = -x_i + (1-\mu_1 x_i)[\gamma_1 \sum_{j=1}^{m} f(y_j)v_{ji} + a_i^k] - (\beta_1 + \alpha_1 x_i)\sum_{j=1}^{m} f(y_j) \quad (2-24)$$

where x_i and y_j are the activation of unit i in F_1 and unit j in F_2 respectively, $f(\ )$ is a nonlinear function defined by

$$f(y_j) = \begin{cases} 1, & \text{if } y_j = \max\{y_k \mid k = 1,2,\ldots,m\} \\ 0, & \text{otherwise.} \end{cases} \quad (2-25)$$

ϵ, μ_1, γ_1, β_1 and α_1 are all positive constant for the regulation of system dynamics. The constraints on their value are discussed in [53]. A similar differential equation which describes the STM activity in F_2 is

$$\epsilon\frac{dy_j}{dt} = -y_j + (1-\mu_2 y_j)[\gamma_2 \sum_{i=1}^{n} S(x_i)w_{ij} + f(y_j)] - (\beta_2 + \alpha_2 y_j)\sum_{\substack{k=1\\k\neq j}}^{n} S(y_k) \quad (2-26)$$

One drawback of the ART network, which is discussed by Lippmann in [6], is that in noise free situations the vigilance parameter can be set to a very low value to discriminate any small difference between input patterns. However in noisy conditions a low vigilance parameter could lead to rapidly increasing number of classes which would exhaust all the units in layer F_2. This problem could be solved by using slow learning rate and an adaptive vigilance parameter approach [6] [54].

2.3 Prewired or Hardwired ANN

Prewired ANN are the networks which have their connection weights predetermined in an initial design process. Thus they are not strictly learning networks. However they can play a significant role by action as preprogramed modules in large learning networks to accelerate the learning and improve the generalization performance. The feature extraction networks used in a Darwin machine is an example [55].

2.3.1 Connectionist Semantic Networks

There has been some research which considers each unit or cell in the ANN as the basic unit for representation of concepts [56] [57]. Usually these cells work in binary mode. Excitation mode or '1' mode represents logical true, and inhibition mode or '0' mode represents logical false. In some research the false is represented by '-1', and '0' is used for 'Unknown' [57]. A simple example showing how this kind of cell or unit works is given in Fig-2.8, where the weights of all the input connections are 1, and the threshold is 0. It uses a step function like that in (2-17) as its activation function. It is obvious the output y is the logic OR of all the input identities. Although these kinds of network lack biological plausibility, they can be used properly for logic inference in semantic networks [58]. A semantic network is a network structure used to express knowledge in terms of concepts, their properties and the relationship between concepts. Each concept is represented by a node and the hierarchical relationship between concepts is represented connecting appropriate concept nodes via **IS-A** or **INSTANCE-OF** links. In [58] six classes of neural units are defined to implement a semantic network. They are enable units, concept units, properties units, concept property binder units, property value binder units and relay units. The high level description of the network is processed by a compiler and a network generator which can generate the topological structure and the weight value of the ANN for that specific task. From this description, it can be seen that this process is very similar to conventional VLSI design. The difference is the connectionist semantic networks are more parallelized.

2.3.2 The Hopfield-Tank Network

Marr's computational theory [59] suggests that to understand the information processing process in brains requires comprehension at several levels. According to

this theory, the information processing process can be understood at three levels. They are computational, algorithm and implementation levels. At the algorithm level, much vision processing especially early vision processing can be characterized as a mathematical transform or minimization problem [60]. The Hopfield-Tank network is an attempt to understand how particular computations can be performed by selecting appropriate patterns of synaptic or weight connectivity in a dynamical ANN [61].

A four unit Hopfield-Tank network is depicted in Fig-2.9. The units or neurons work in analog mode and are a simplified model of biological neurons. The activation of the units can be described by the differential equation

$$C_i \frac{dx_i}{dt} = \sum_{j=1}^{N} w_{ji} g_j(x_j) - \frac{u_i}{R_i} + I_i \qquad (2-27)$$

where C_i is the characteristic capacitance of cell i, R_i is the equivalent membrane resistance (or leakage resistance), w_{ji} is the conductance of connection from cell j to cell i, x_i is the electric potential of cell i, $g_j(\ )$ are sigmoid functions and I_i are external stimulus current. So the function of a cell can be summarised as charging the characteristic capacitance C_i with the sum of postsynaptic currents induced in cell i by presynaptic activity in cell j and the leakage current due to R_i and the input current I_i from other sources external to the network.

An energy function can be defined on the network as [62]

$$E = -\frac{1}{2} \sum_{i,j} w_{ij} y_i y_j + \sum_i \frac{1}{R_i} \int_0^{y_i} g_i^{-1}(v) dv - \sum_i I_i y_i \qquad (2-28)$$

where $y_i = g_i(x_i)$. It can be proved that

$$\frac{dE}{dt} \leq 0; \quad \frac{dE}{dt} = 0 \rightarrow \frac{dy_i}{dt} = 0 \quad for\ all\ i \qquad (2-29)$$

So the network is a stable system. If the sigmoid function $g_i(\ )$ has a steep rise (or high gain), then the E function defined in (2-28) can be approximated by [62]

$$E = -\frac{1}{2}\sum_{i,j} w_{ij} y_i y_j - \sum_i I_i y_i \qquad (2-30)$$

Thus the operation of the network is very similar to binary networks. This feature has been explored successfully in the application of Hopfield-Tank network to various optimization problems [63] [61].

For high gain function $g_i(\ )$, its shape is very close to a step function. So the output y_i is usually clamped to take only two states. They can be regarded as '1' and '0' states like those in digital circuits. The states of the network can therefore be used to represent binary numbers. Thus the network can be used in implementing an A/D converter [63]. If the network has n cells, and external analog input is S, then an n-bit A/D converter can be designed by minimizing the energy function

$$E = \frac{1}{2}(S - \sum_{i=1}^{n} y_i 2^i)^2 \qquad (2-31)$$

To help clamp the states of the network into corners of the state space, or to push y_i into '1' or '0' states, a second term

$$-\frac{1}{2}\sum_{i=1}^{n}(2^i)^2[y_i(y_i-1)] \qquad (2-32)$$

is added to the E defined in (2-31). So the final energy function is

$$E = -\frac{1}{2}\sum_{\substack{i=1,j=1\\ i\neq j}} (-2^{i+j}) y_i y_j - \sum_{i=1}^{n}(-2^{(2i-1)} + 2^i S) y_i \qquad (2-33)$$

If this expression is compared with (2-30), it follows that

$$w_{ij} = -2^{i+j}$$

$$I_i = (-2^{(2i-1)} + 2^i S) \qquad (2-34)$$

If a Hopfield-Tank network is assigned the connection weights given by (2-34), the operation of the network would lead to the minimization of the energy function defined in (2-33), Thus under an analog input excitation S, the stable state of the network $[y_1, y_2, \ldots, y_n]$ would be a binary representation of S.

However the Hopfield-Tank network has a local minima problem. For hardware implementation this can be a serious problem. Adding correction current I_{ic} into the cells could help to eliminate this problem [64].

2.3.3 Silicon Neural Systems

The silicon neural systems here are a class of silicon artificial neural networks which are based on biological information processing mechanisms and implemented by analog VLSI technology [65], like electronic cochlea [66] and silicon retina [67] for example. These systems do not copy biological features like anatomy structure directly, instead they imitate the computation functions of biological systems by electronic circuits. The motion computing network is a typical example [68]. There are two methods of computing motion for vision systems. One is the intensity-based scheme and the other is a token identification scheme. Both schemes coexist in biological systems [68]. For the intensity-based scheme, the computation of motion can be abstracted as the minimization of

$$E(u,v) = \int\int (I_x u + I_y v + I_t)^2 + \lambda[(\frac{\partial u}{\partial x})^2 + (\frac{\partial u}{\partial y})^2 + (\frac{\partial v}{\partial x})^2 + (\frac{\partial v}{\partial y})^2]dxdy \quad (2-35)$$

where (u, v) represent the velocity field, $I(x, y, t)$ is light intensity and λ is the regulation parameter. This minimization problem is equivalent to solving a set of linear equations [68]

$$\begin{aligned} I_x^2 u + I_x I_y v - \lambda \nabla^2 u + I_x I_t &= 0 \\ I_x I_y u + I_y^2 v - \lambda \nabla^2 v + I_y I_t &= 0 \end{aligned} \quad (2-36)$$

and (2-36) can be discretized into

$$I_{xij}^2 u_{ij} + I_{xij} I_{yij} v_{ij} - \lambda(u_{i+1,j} + u_{i,j+1} - 4u_{ij} + u_{i-1,j} + u_{i,j-1}) + I_{xij} I_{tij} = 0$$

$$I_{xij} I_{yij} u_{ij} + I_{yij}^2 v_{ij} - \lambda(v_{i+1,j} + v_{i,j+1} - 4v_{ij} + v_{i-1,j} + v_{i,j-1}) + I_{yij} I_{tij} = 0 \qquad (2-37)$$

where (i, j) is the index of a lattice grid. The electric current in a lattice electronic resistive network can be calculated by a similar set of equations if Kirchhoff's current law is used. So (2-37) can be solved by measuring the current in an electronic resistive network which is under a proper external excitation [68].

These silicon networks basically follow Marr's computation theory philosophy [59]. They are an attempt to imitate the computation functions of biological systems instead of their implementation detail. They are more like a functional model of neural systems.

In this chapter we have surveyed briefly some ANN. In the following chapters, we discuss some theoretical properties of ANN and their potential application in engineering.

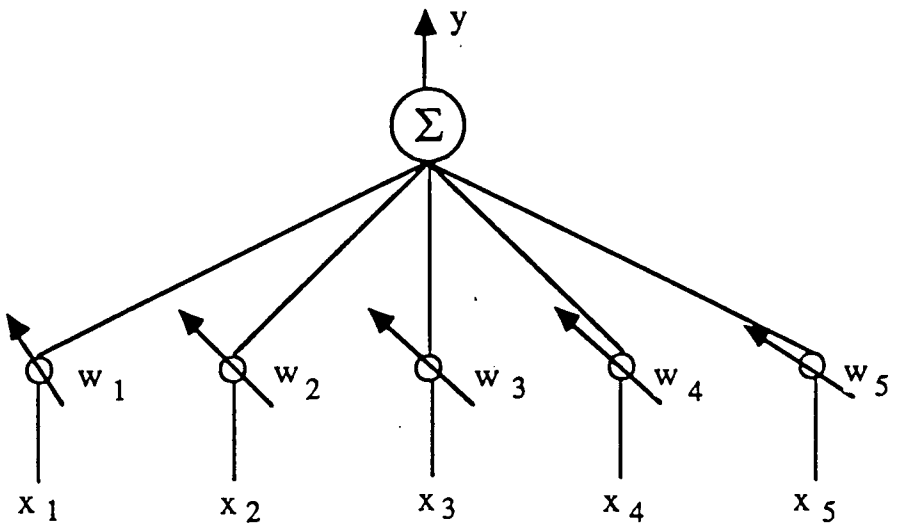


Fig-2.1 Adaptive Element

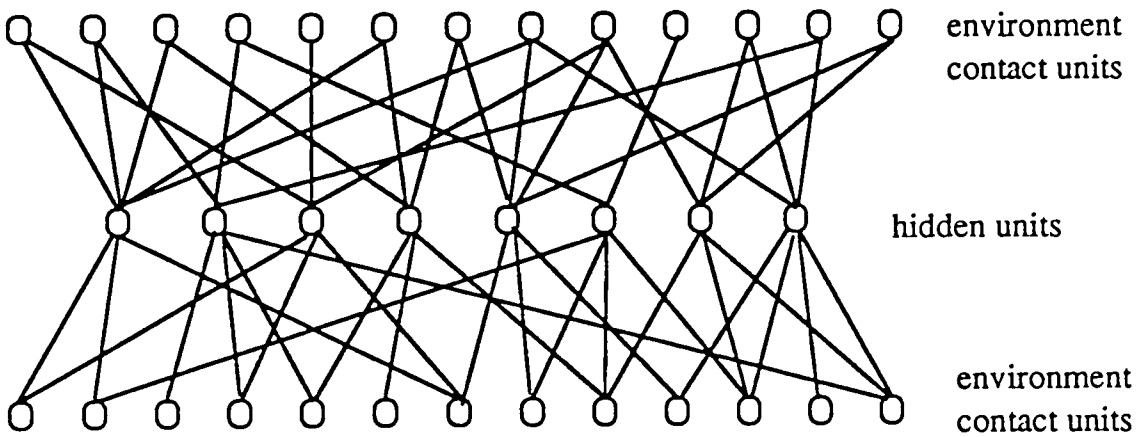


Fig-2.2 A Boltzmann Machine

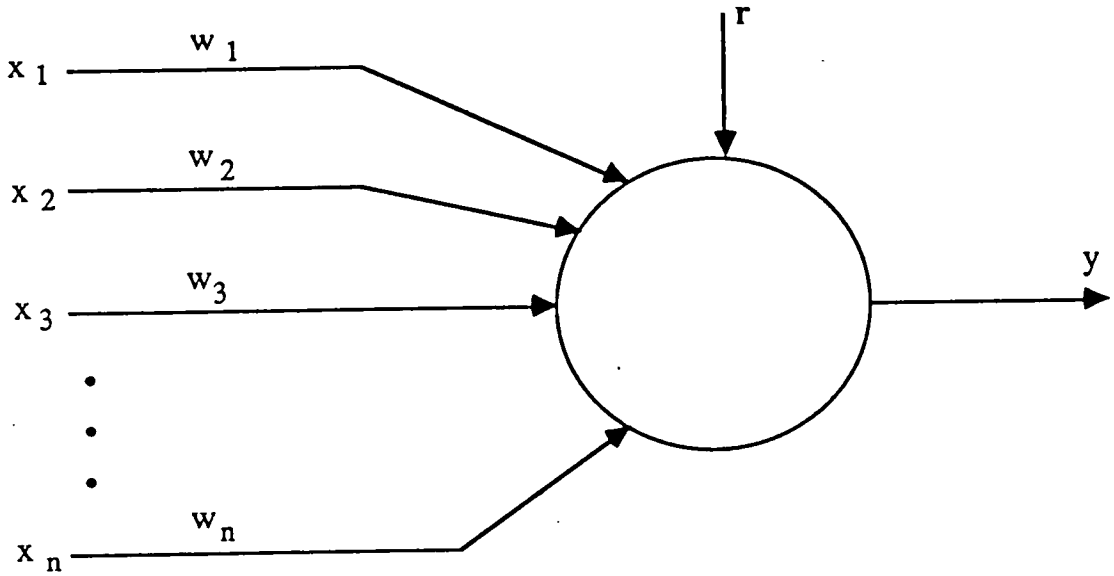


Fig-2.3 An ARP learning Element

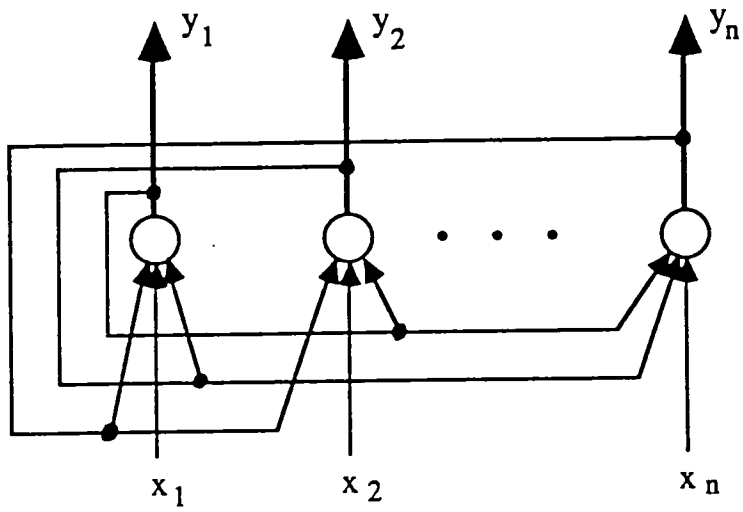


Fig-2.4 Hopfield Associative Memory Network

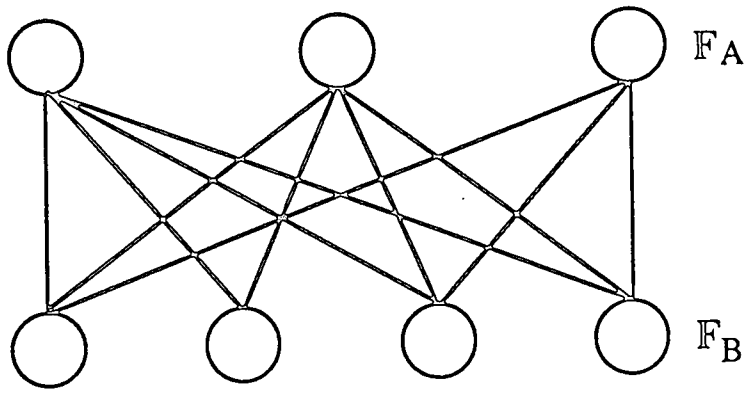


Fig-2.5 A BAM Network

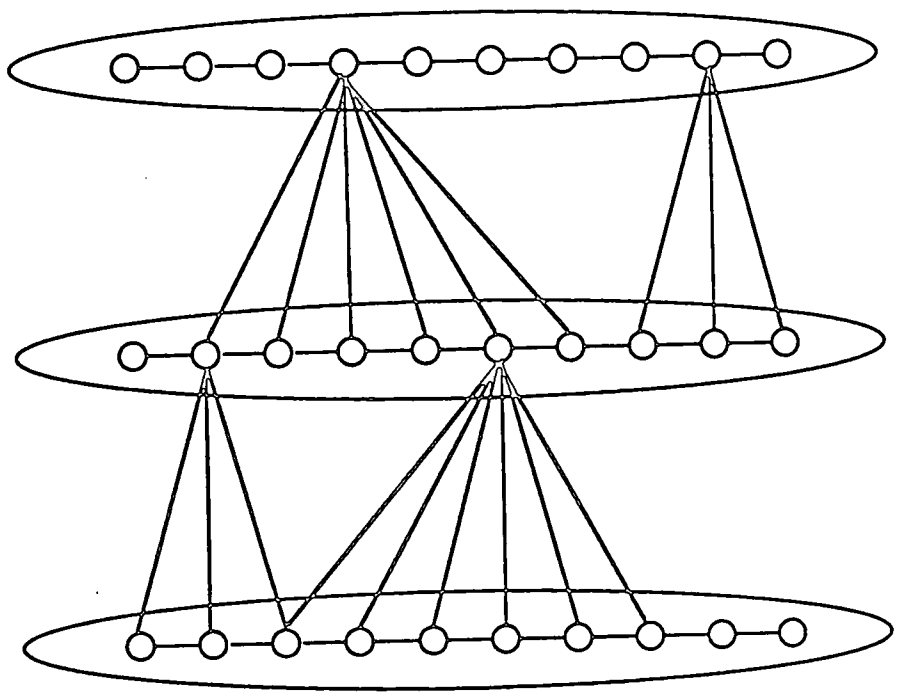


Fig-2.6 Competitive Learning

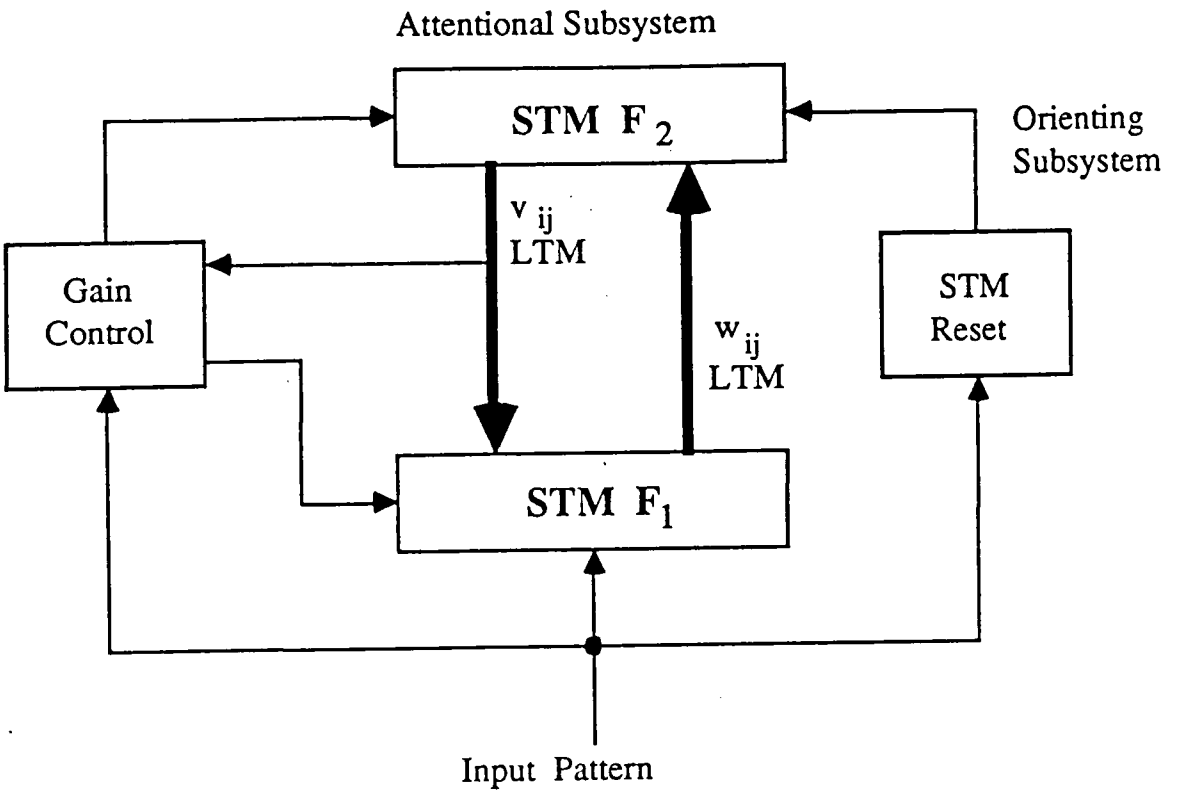


Fig-2.7 ART1 Network

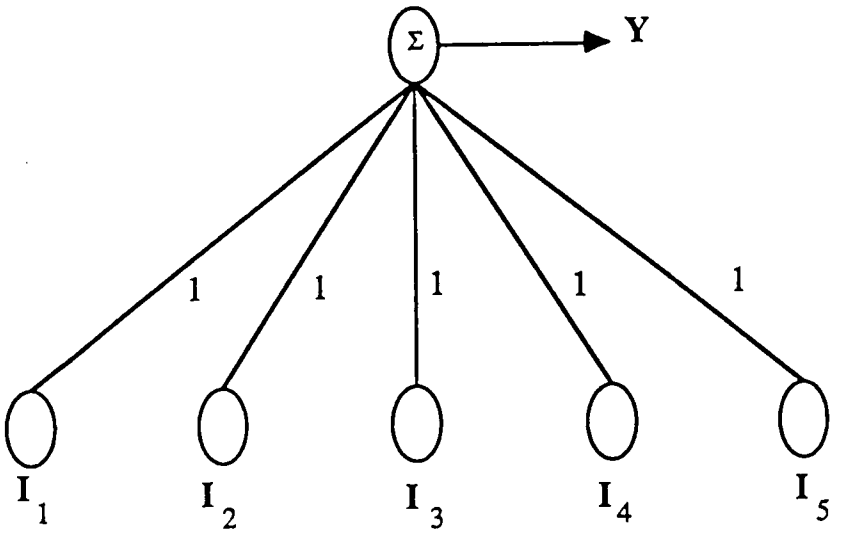


Fig-2.8 Logical OR Unit Implemented by Neural Network

▬ represent the conductance of the connection

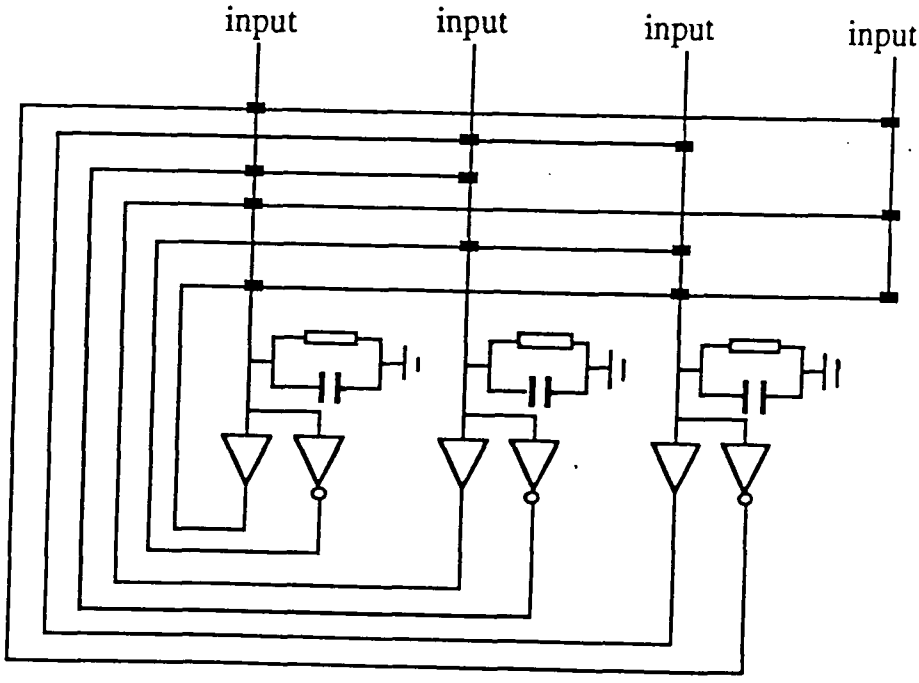


Fig-2.9 Hopfield-Tank Network

Chapter Three
MLP and Back-Propagation Algorithm

Among the existing ANN, the Multi-Layer-Perceptron (MLP) using a Back-Propagation learning algorithm is one of the most widely used networks because of its simplicity and powerful representation ability. The single layer perceptron [69] has a limited representation ability. For example, it can not implement the XOR function. This representation limitation has been thoroughly discussed by Minsky and Papert in the late 1960s [16]. Unlike the single layer perceptron, MLP networks can implement any complicated function due to the additional hidden layer. In the following sections, the Back-Propagation learning algorithm and some techniques for improving its learning speed are discussed.

3.1 Delta Rule and Back-Propagation

An MLP network is depicted in Fig-3.1. It is a layered and feed-forward network. The output from a unit can only feed into a unit in an adjacent layer. Recurrent connections and connections within the same layer are not allowed.

The input to a unit in the network is a weighted sum of the outputs of units in the previous layer to which the unit is connected. Let x_i^j represent the input to the unit i in layer j, w_{ij}^k stand for the weight of the connection from unit i in layer k to unit j in layer $k+1$, y_i^j is the output of unit i in layer j and θ_i^j is the threshold (or bias) of unit i in layer j. Then the activation of units can be described by

$$x_j^{k+1} = \sum_i w_{ij}^k y_i^k \qquad (3-1)$$

and

$$y_i^j = f(x_i^j) = \frac{1}{1 + exp(-\frac{x-\theta}{T})} \qquad (3-2)$$

The activation function f( ) can take other forms. For example, in applications where negative output is needed, it can be of the form

$$f(x) = \frac{1 - exp(-\frac{x-\theta}{T})}{1 + exp(-\frac{x-\theta}{T})} \qquad (3-3)$$

For MLP networks, there are two modes of operation during the training or learning phase. They are feed-forward computation and the weight updating operation. In feed-forward computation, when an input pattern or vector is presented to the input layer, the units in the next layer use the weighted sum of inputs and the activation function defined in (3-2) or (3-3) to calculate their outputs. These outputs are passed forward for computation in the next layer until the output layer is reached. During the weight updating operation, an error signal which is based on the discrepancy between the desired response and the actual output of the network is backpropagated through the network for the updating of weights. Obviously this Back-Propagation algorithm is a form of spervised learning. During the recall process, only feed-forward computation is involved. The derivation of the Back-Propagation algorithm is presented in the following paragraphs.

Basically Back-Propagation is a gradient decent algorithm. When an input pattern is presented to the input layer of the network, a corresponding output will be produced by feed-forward computation. This is called a forward pass. The discrepancy between the desired or target output d_i and actual output y_i can be measured by

$$E = \frac{1}{2}\sum_j (d_j - y_j)^2 \qquad (3-4)$$

Let the number of training samples be N, the total error is

$$E_{total} = \sum_{k=1}^{N} E_k \qquad (3-5)$$

where E_k denotes the discrepancy error of the kth pattern calculated by (3-4).

The objective of the training phase is to minimize the E_{total} defined in (3-5). A straight forward way of mininization is to use a gradient descent (hill-climbing) algorithm, that is to adjust the weights according to an amount proportional to the derivative

$$\Delta w_{ij}^k = -\eta \frac{\partial E}{\partial w_{ij}^k} \tag{3-6}$$

where η is the step size, which is used to control the convergence. The convergence can be improved significantly by the inclusion of a momentum term. Actually this is a first order low pass filter which smooths out fast changes in Δw_{ij}^k.

$$\Delta w_{ij}^k(t) = -\eta \frac{\partial E}{\partial w_{ij}^k(t)} + \alpha \Delta w_{ij}^k(t-1) \tag{3-7}$$

α is a small positive real number.

In the strict sense gradient descent algorithm, the E in (3-6) and (3-7) should be E_{total}, that is the updating of weights would take place once every pass through all the training samples. This is called batch mode Back-Propagation. However a more common practice is to update weights for every presentation of an input training sample. In this case the E defined in (3-4) can be used to replace E in (3-6) and (3-7) for calculation. This is a wide sense gradient decent algorithm.

For a single layer perceptron, the output is given by

$$y_j = f(\sum_{i=1}^{n-1} w_i I_i + \theta_j) = f(s) \tag{3-8}$$

As the threshold θ_j can be regarded as an input from an always active unit through a connection with weight θ_j, (3-8) can be rewritten as

$$y_j = f(\sum_{i=1}^{n} w_i I_i) = f(s) \tag{3-9}$$

If the desired output is d_j, From (3-4), it is obvious that

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial w_i} \\ &= -(d_j - y_j) f'(s) I_i \end{aligned} \tag{3-10}$$

Let

$$\delta_j = (d_j - y_j)$$

then the adjustment of weights is given by

$$\begin{aligned} \Delta w_i &= -\eta \frac{\partial E}{\partial w_i} \\ &= \delta_j f'(s) I_i \end{aligned} \tag{3-11}$$

This weight updating scheme is sometimes called the Delta rule. Back-Propagation is also refered to as the generalized Delta rule.

For Back-Propagation in an MLP network, the weight updating for the connection to the output layer is straight forward. If the MLP network has N layers and function f( ) defined in (3-2) is used as an activation function, then from (3-4)

$$\frac{\partial E}{\partial y_j^N} = -(d_j - y_j^N) \tag{3-12}$$

where the superscript N means the output is from layer N and from (3-2)

$$\frac{\partial y_j^N}{\partial x_j^N} = f'(x_j^N) = y_j^N(1 - y_j^N) \tag{3-13}$$

From equation (3-1) it is clear that

$$\frac{\partial x_j^N}{\partial w_{ij}^{N-1}} = y_i^{N-1} \tag{3-14}$$

So

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}^{N-1}} &= \frac{\partial E}{\partial y_j^N} \frac{\partial y_j^N}{\partial x_j^N} \frac{\partial x_j^N}{\partial w_{ij}^N} \\ &= -(d_j - y_j^N) y_j^N (1 - y_j^N) y_i^{N-1} \end{aligned} \tag{3-15}$$

Let

$$\delta_j^{N-1} = (d_j - y_j^N) y_j^N (1 - y_j^N) \tag{3-16}$$

then

$$\frac{\partial E}{\partial w_{ij}^{N-1}} = -\delta_j^{N-1} y_i^{N-1} \tag{3-17}$$

So from (3-7) and (3-17) we have

$$w_{ij}^{N-1}(t+1) = w_{ij}^{N-1}(t) + \eta \delta_j^{N-1} y_i^{N-1} + \alpha[w_{ij}^{N-1}(t) - w_{ij}^{N-1}(t-1)] \tag{3-18}$$

This is the weight updating formula for the connection to the output layer. Note that the delta in (3-16) can be rewritten as

$$\delta_j^{N-1} = -\frac{\partial E}{\partial y_j^N}\frac{\partial y_j^N}{\partial x_j^N} = -\frac{\partial E}{\partial x_j^N} \tag{3-19}$$

then generally for weight w_{ij}^k we have

$$\begin{aligned}
\frac{\partial E}{\partial w_{ij}^k} &= \frac{\partial E}{\partial y_j^{k+1}}\frac{\partial y_j^{k+1}}{\partial x_j^{k+1}}\frac{\partial x_j^{k+1}}{\partial w_{ij}^k} \\
&= \frac{\partial E}{\partial y_j^{k+1}} y_j^{k+1}(1-y_j^{k+1}) y_i^k \\
&= y_j^{k+1}(1-y_j^{k+1}) y_i^k \sum_l \frac{\partial E}{\partial x_l^{k+2}}\frac{\partial x_l^{k+2}}{\partial y_j^{k+1}} \\
&= y_i^k y_j^{k+1}(1-y_j^{k+1}) \sum_l \frac{\partial E}{\partial x_l^{k+2}} w_{jl}^{k+1} \\
&= -y_i^k y_j^{k+1}(1-y_j^{k+1}) \sum_l w_{jl}^{k+1}\delta_l^{k+1}
\end{aligned} \tag{3-20}$$

let

$$\delta_j^k = y_j^{k+1}(1-y_j^{k+1}) \sum_l w_{jl}^{k+1}\delta_l^{k+1} \tag{3-21}$$

then

$$\frac{\partial E}{\partial w_{ij}^k} = -\delta_j^k y_i^k \tag{3-22}$$

This equation is similar to (3-17). If we combine equations (3-18) and (3-21), we obtain the complete Back-Propagation algorithm. The delta's are propagated backward from the output layer by equation (3-21). This is why the algorithm is called Back-propagation algorithm. The initial delta is calculated using (3-16).

The program structure for implementation of the Back-Propagation algorithm on MLP networks is shown in Fig-3.2. The initializing program in box-1 is used to initiate weights and thresholds of MLP networks with stochastic or deterministic techniques. A common practice is to assign a small value which is taken from a random number generator to every weight and threshold. The program in box-2 is used to set up a training sample set or learning sample set. The main program in box-3 is the implementation of the Back-Propagation algorithm. Its structure is depicted in Fig-3.3. A complete pass through the loop is called a cycle. The weight updating occurs in every cycle. However in the strict sense gradient descent algorithm, $\frac{\partial E}{\partial w_{ij}}$ are accumulated over a pass through all the samples in the training set, and then weights are updated. The program in box-4 is for display purposes.

Input X_1	Input X_2	Output f
0	0	0
0	1	1
1	0	1
1	1	0

Table-3.1 XOR function

To demonstrate the learning ability of the MLP network with a Back-Propagation algorithm, the above simulation program is used to train an 2-2-1 MLP network to implement a XOR function, which is not possible by a single layer perceptron. The XOR function can be summarized by Table-3.1 The implementation MLP network is shown in Fig-3.4. The displayed value of weights and thresholds are obtained by the Back-Propagation algorithm. Further examples which demonstrate the learning ability of MLP networks with Back-Propagation algorithm can be found in [70] [19]. Although the MLP network has strong representation ability,

the Back-Propagation algorithm is very slow. In the next section the convergence of the Back-Propagation is discussed.

3.2 The Convergence of Back-Propagation

As the back-propagation algorithm is a kind of gradient descent algorithm, the error surfaces for learning problems frequently possess some geometric properties that makes the algorithm slow to converge. The stepsize of the algorithm is sensitive to the local shape and curvature of the error surfaces. For example, a small stepsize will make the algorithm take a very long time to cross a long flat slope. On the other hand, a large stepsize will cause the iteration process to bounce between the two opposite sides of a valley rather than following the contour of its bottom. Even if a satisfactory stepsize for one stage of the learning process is found, this does not ensure it will be appropriate for any other stage of the same learning process. On the other hand, the premature saturation of the network units also causes problems for the convergence of the algorithm.

There has been some research on improving the convergence speed of the Back-Propagation algorithm, such as that mentioned in [71] [72] [73]. In [71] the authors suggested Conjugate gradients, the Quasi-Newton algorithm and other more sophisticated algorithms. The conjugate gradient with linear search is also reported in [74]. They are also called second order methods, and these algorithms are more computationally expensive, especially when the scale of the problem is large, so that in many cases it is impractical to use them. In order to reduce the computation cost of the second order method, a kind of approximation technique has been introduced into Newton's algorithm[72]. The authors used a diagonal matrix to approximate the Hessian matrix. This makes it possible to derive a back propagation algorithm for the second order derivatives in a similar manner to the first order derivatives.

But the applicability of this new algorithm depends on how well the diagonal Hessian approximation models the true Hessian[72]. Only when the effects of weights on the output are uncoupled or nearly uncoupled, can the diagonal Hessian represent a good approximation. In addition the learning parameters are more critical in obtaining reasonable behaviour with this Newton-like algorithm than with the back-propagation algorithm [72]. Our simulation also confirm this point. Another attempt to use a second order method to improve the convergence property of the back-propagation algorithm was introduced in [73], which is called Quickprop. It uses the difference between two successive $\frac{\partial E}{\partial w}$ as a measure of the change of curvature and uses this information to change the stepsize of the algorithm. E is the output error function, and w represent weights. Using this method a significant improvement in convergence speed has been reported in [73].

In [75] another kind of adaptive stepsize algorithm was introduced. According to this algorithm, if an update of weights results in reduced total error, the stepsize is increased by a factor $\phi > 1$ for the next iteration. If a step produces a network with a total error more than a few percent above the previous value, all changes to the weights are rejected, the stepsize is reduced by a factor $\beta < 1$, the momentum term is set to zero, and the step is repeated. When a successful step is then taken, the momentum term is reset.

As is well known in adaptive signal processing theory, the direction of the negative gradient vector may not point directly towards the minimum of the error surface. In adaptive filter theory, this kind of bias can be measured by the ratio of the maximum eigenvalue and the minimum eigenvalue of the auto-correlation matrix[76]. Recently an adaptive stepsize algorithm which gives every weight a stepsize which can adapt separately has been proposed[77]. This is only a rough approximation, as it will be noted that these stepsizes adapt in the direction of each

weight rather than on the eigenvector direction as required[76][77].

As mentioned in the previous section, the update of weights can take place after presenting all the training samples to the network or after every presentation of a training sample, these methods are called batch mode back-propagation and online back-propagation respectively. Generally speaking, online back-propagation algorithms converge faster than the batch mode back-propagation[72][73], and batch mode back-propagation is more likely to fail to converge on a large training sample set[78]. In the following section, two stepsize variation techniques are introduced for acceleration of the online Back-Propagation algorithm. Compared with previous algorithms, they are more simple to implement.

3.3 Acceleration of Back-Propagation

3.3.1 Adaptive Stepsize Technique

In designing an appropriate algorithm the following factors should be considered. First the momentum term cannot be set to zero, as the update occurs for every presentation of a new training sample. If the momentum term is set to zero, there exists a risk of losing past experience. Generally speaking. a large training sample set requires a large η value (η is the stepsize for the momentum). This fact has been confirmed by computer simulation[79]. Thus the adaption is restricted to the gradient term. We used the following form of adaptive stepsize algorithm:

$$\alpha(t) = \alpha(t-1)(1 - f(t)\sqrt{E(t)}\ ) \qquad (3-23.a)$$

$$f(t) = u_1 f(t-1) + u_2 \Delta E(t) \qquad (3-23.b)$$

$$\Delta E(t) = E(t) - E(t-1) \qquad (3-23.c)$$

$\alpha(t)$ is the stepsize for the gradient term in the update formula in the back-propagation algorithm. It is the stepsize at time t. E(t) is the summation of

squared errors between the desired output and the actual output at time t. It can be calculated as

$$E = \frac{1}{2}\sum_{k=1}^{N}\sum_{i=1}^{p}(d_i^k - o_i^k)^2 \qquad (3-24)$$

$\Delta E(t)$ is the decrement of the E(t). f(t) is a filtered version of $\Delta E(t)$. Actually (3-23.b) is a first order low-pass recursive filter, which can smooth the significant changes in $\Delta E(t)$, making the algorithm more stable. u_1 and u_2 are the parameters used to control the adaptation. For small u_1 and big u_2, the adaptation is fast, but it is also more likely to be trapped in oscillation. For big u_1 and small u_2, the adaptation is slow, but it is more stable. Thus the parameter selection involves a trade off. In our simulation, we used $u_1 = 0.9$ and $u_2 = 0.3$. The term $(1 - f(t)\sqrt{E(t)}\ )$ also controls the adaptation of the stepsize. If f(t) is positive, that means the tendency of E(t) in the near past is to increase, so $1 - f(t)\sqrt{E(t)} < 1$, the stepsize will be decreased. A similar analysis shows that if the tendency of E(t) is to decrease, the stepsize will be increased. When the E(t) is very small, that is the network has almost learned, the adaption will be very weak, which stablizes the algorithm. The square root is used as compensation, it can amplify the small E(t) to avoid the premature termination of adaptation.

Before we start to discuss the simulation of the adaptive stepsize algorithm, we give a brief description about two commonly used benchmark test problems for ANN.

Parity Problem : The parity function is a generalization of XOR function which is described in section 3.1. It only considers a binary input. For a N-bit parity function, when there odd number of '1' in the input vector, the output is '1', otherwise the output is '0'. As mentioned in the previous discussion the XOR function cannot be realized by a single layer perceptron, but is amenable to MLP networks,

so the parity functions are commonly used as a test problem for ANN.

Compression Encoding Problem : The ANN used for this kind of encoding usually takes the structure of N-M-N. It means both input and output layers have N units and the hidden layer has M units. Usually $M < N$. The network can only accept binary input and the output also takes binary form. From information theory we know that if there are S patterns, then binary codes of length $\log_2 S$ can code all these patterns. This encoding problem is used to test whether the ANN can learn to compress a set of redundant N-nit binary codes to more compacted M-bit codes.

The simulation results on adaptive stepsize Back-Propagation algorithm are shown in Fig-3.5, Fig-3.6, Fig-3.7 and Fig-3.8. In the figures shown, the E defined in (3-24) are plotted as a function of iteration times for different learning problems. They are called learning curves, and can be used to evaluate the convergence property of the learning algorithm. Fig-3.5 shows comparative simulation results of the non-adaptive back-propagation algorithm and the adaptive algorithm for the XOR problem. Fig-3.6 shows the comparision on the 4-2-4 encoder problem. Fig-3.7 and Fig-3.8 are for the 4-bit parity problem implemented with 4-4-1 MLP network. All broken lines represent learning curves for the non-adaptive algorithm, while solid lines are for the adaptive algorithm. It is clear the adaptive stepsize has improved the convergence speed, just as we expected. However, it will be noted that the improvement for a complex problem, like the 4-bit parity problem, is more impressive than that for the simpler problem, like the XOR problem or the 4-2-4 encoder problem. The reason may be that since adaptation is a dynamic process, it needs a finite time to be effective. For simple problems, the learning process is very short, and the adaptation process has insufficient time to be significant. Thus there is only a small effect of adaption on simple learning problems.

3.3.2 Differential Stepsize Back-Propagation

Although the adaptive stepsize back-propagation algorithm has improved the learning speed to some degree, it can not cope with the premature saturation of the network units. It has been noted in the simulations that MLP neural nets are often trapped in a very flat valley in which the convergence speed is very slow. This corresponds to the flat line intervals on the learning curves of Fig-3.7 and Fig-3.8. It should be noticed that this cannot be solved by an adaptive stepsize technique, because the reason for this phenomenon is that the absolute value of weights are growing so fast as to make the units, especially hidden units, prematurely saturated. There is a term like s(1-s) in the update formula for the back-propagation algorithm, in which s is the output state of the unit. It is quite clear that if s is close to 1 or 0, whichever output is desirable, almost no update will be passed backward through that unit. This kind of phenomenon is also known as the flat spot[73]. In [73] the author proposed to change the sigmoid-prime function s(1-s) to s(1-s)+0.1, so it can avoid the flat spot. But according to our simulations, this change often causes the weights to grow so fast as to lead to floating point overflow on the digital computer. Although some weight-decay term may be used to counteract this[72], it makes the algorithm more complex. A simple method can be used to cope with the flat spot.

To circumvent the flat spot, the term s(1-s) is removed from the update formula for the output layer, and the stepsize for the update of weights between the hidden layer and the input layer is set smaller than that for the weights between the upper layers. If denote the stepsize for the update of weights between the output layer and the hidden layer as α_2, and the stepsize for the update of weights between the hidden layer and the input layer as α_1, then $\alpha_2 > \alpha_1$. This is called the differential stepsize back-propagation algorithm(DSBP). In our simulation, the learning parameters are set to $\alpha_1 = 0.1\alpha_2$. The simulation results are shown in Fig-3.9, Fig-3.10 and Fig-

3.11, and it is very clear the convergence speed is improved considerably.

In [73] the Quickprop algorithm was claimed to be the fastest learning algorithm among the existing algorithms. In order to compare the DSBP with the Quickprop, the simulation has been repeated 30 times on the 10-5-10 encoder problem. The termination condition for the simulation is that the discrepancy between the desired output and the actual output for every output unit and every training sample is less than 0.1. The average training time in terms of iterations for this problem by DSBP is 23.5, with a standard derivation of 3.27. This is only marginally slower than the Quickprop algorithm, for which the average training time is 22.1. However although the Quickprop plus a hyperbolic arctan error function algorithm can reach the same solution with an average training time of 14.01, it is much more complex than DSBP, and a weight-decay term is needed. The results for the simple DSBP algorithm represent a considerable improvement on the standard back-propagation algorithm, which gave an average training time of 129 iterations.

From the above discussion, it can be seen that the adaptive stepsize technique can improve the convergence performance of the Back-Propagation algorithm. The simulation results presented also suggest the inprovement of performance is more obvious on a large training sample set problem than that on small training sample set problems. Thus it is reasonable to speculate that there would be more potential for using adaptive stepsize technique in large scale application problems, like Net-Talk [80]. The simulation results also show that the DSBP method is effective in circumventing the premature saturation or flat spot under some circumstances. The improvement in convergence speed is significant.

Numerical optimization is still based more on empirical experience rather than on rigorous theoretical analysis. It has been pointed out by Powell [81] that while

the vast majority of theoretical questions related to the performance of optimization algorithms in practice are unanswered, it would be both unrealistic and counter-productive to expect each new algorithm to be justified theoreticallly.

As the Back-Propagation algorithm is basically a numerical optimization algorithm, the measurement of its performance has to largely depend on simulation test on some bench mark problems. Many algorithms have been suggested to accelerate the learning speed of Back-Propagation [82]. Different algorithms may show advantages for specific applications. It would be extremely difficult if not impossible to analyze their performance in a mathematically rigorous way. At this stage a realistic strategy for selecting an appropriate algorithm for a specific problem can only be based on simulation study on similar problems and empirical knowledge.

In the next chapter, we will discuss another important aspect of ANN, the generalization problem.

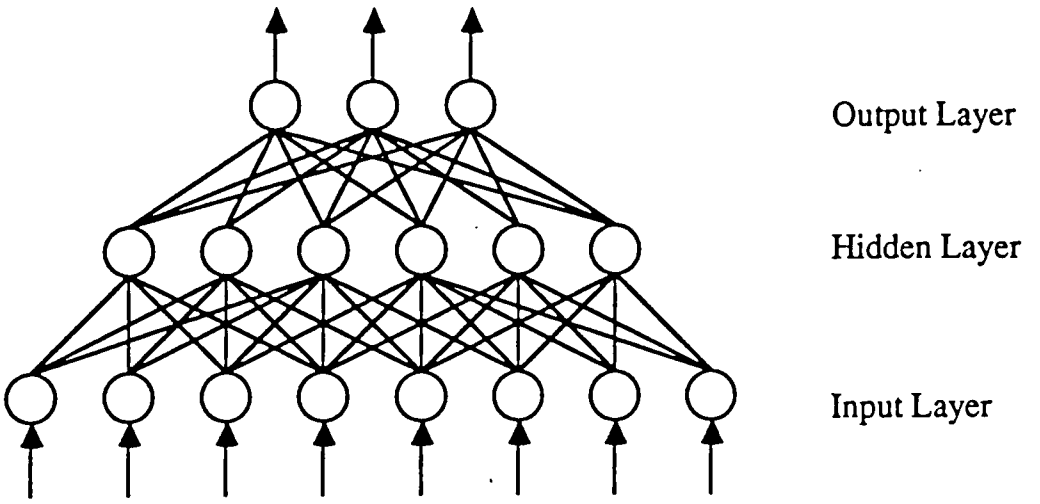


Fig-3.1 A MLP Network

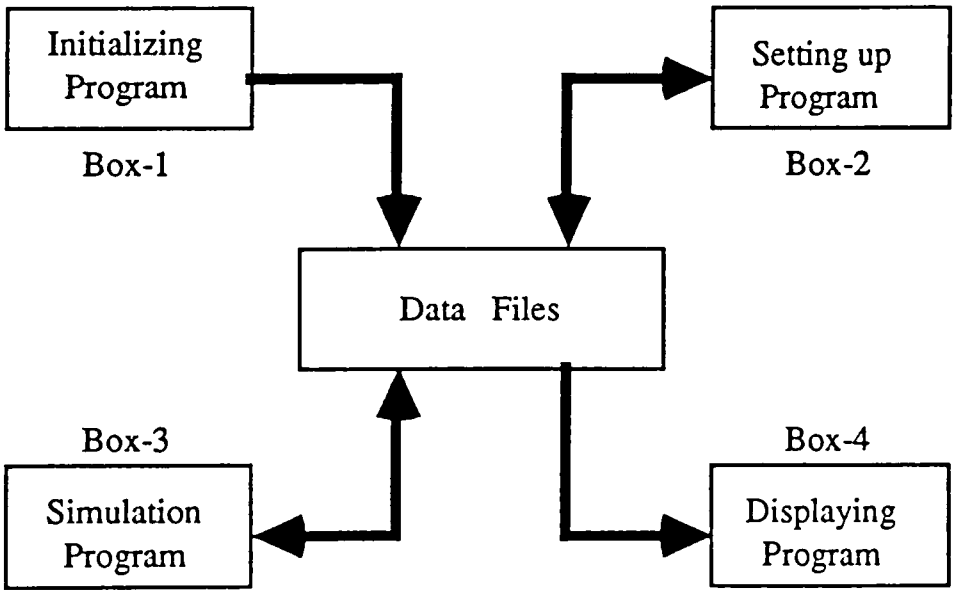


Fig-3.2 Simulation Tool Structure

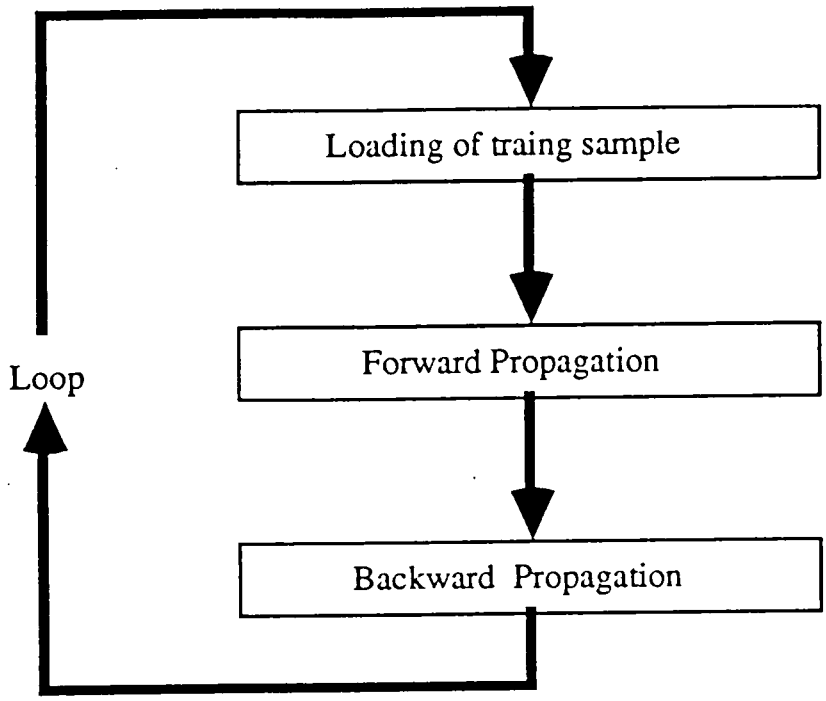


Fig-3.3 Structure of Main Program

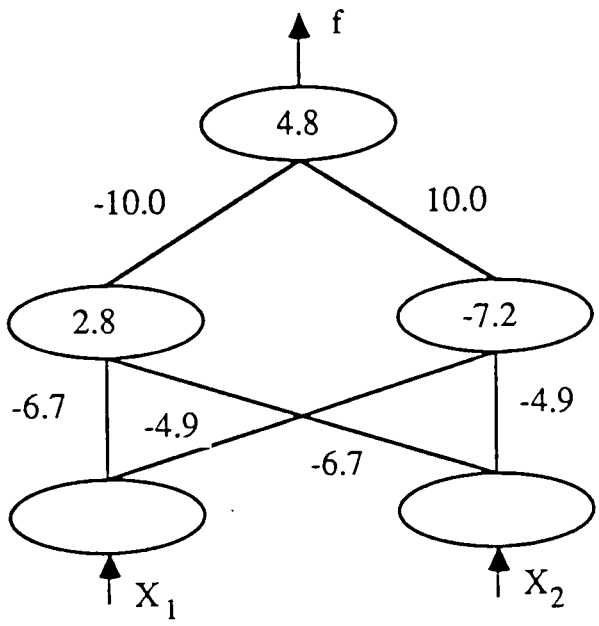


Fig-3.4 MLP XOR Network

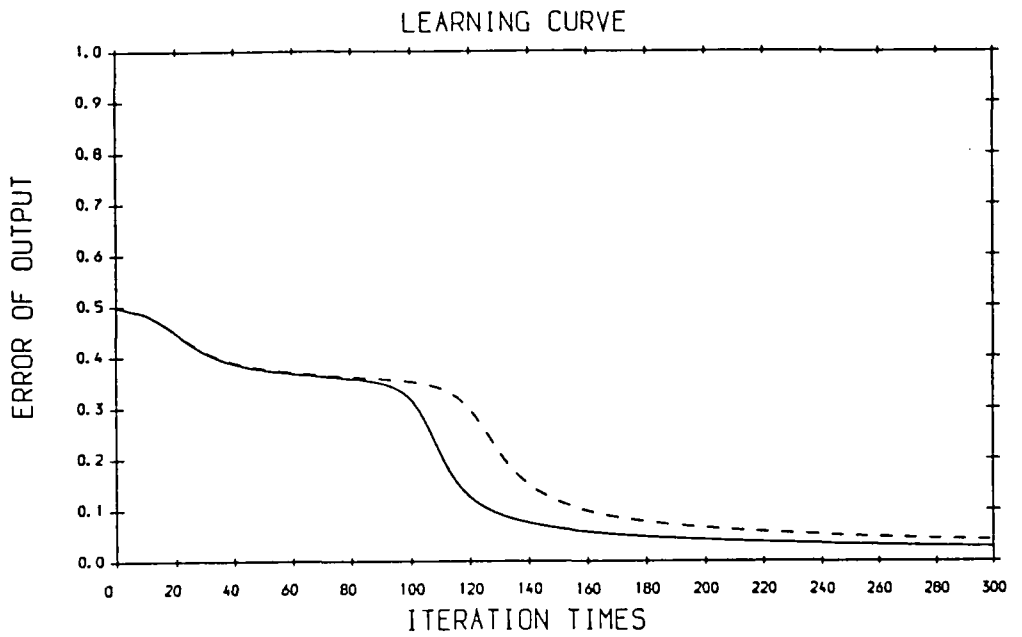


Fig-3.5

Learning curves for the XOR problem. Broken line stands for the learning curve of non-adaptive algorithm.

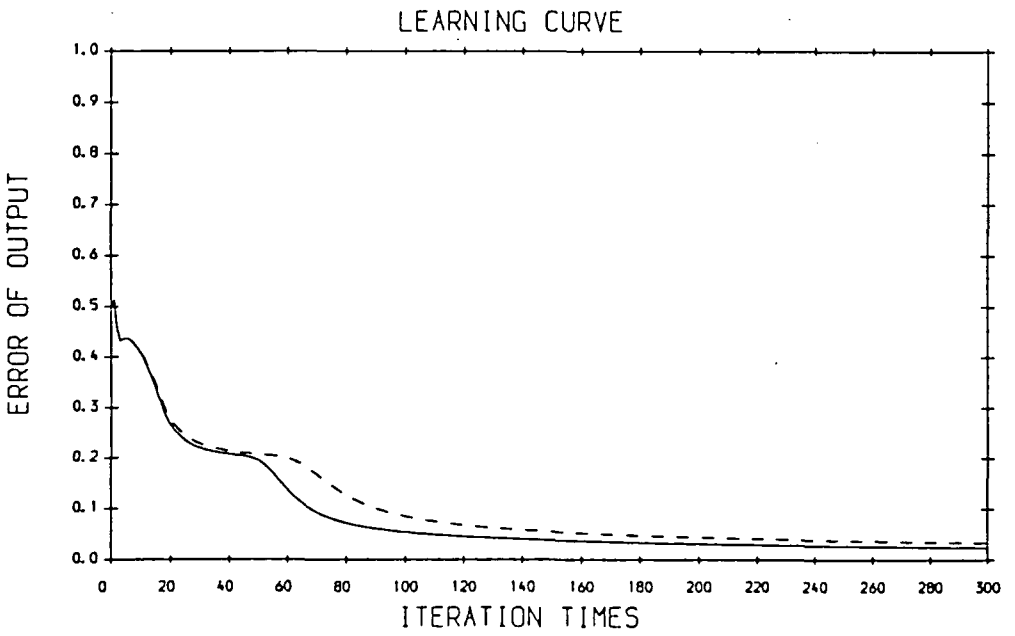


Fig-3.6

Learning curves for the 4-2-4 encoder problem. Broken line stands for the learning curve of non-adaptive algorithm.

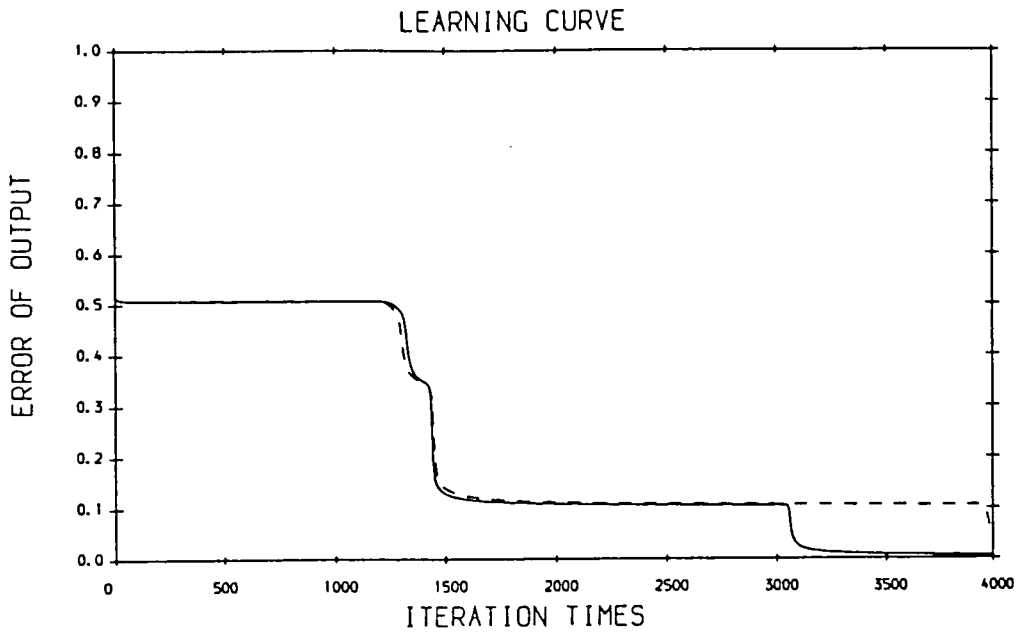


Fig-3.7

Learning curves for the 4-4-1 parity problem. Broken line stands for the learning curve of non-adaptive algorithm.

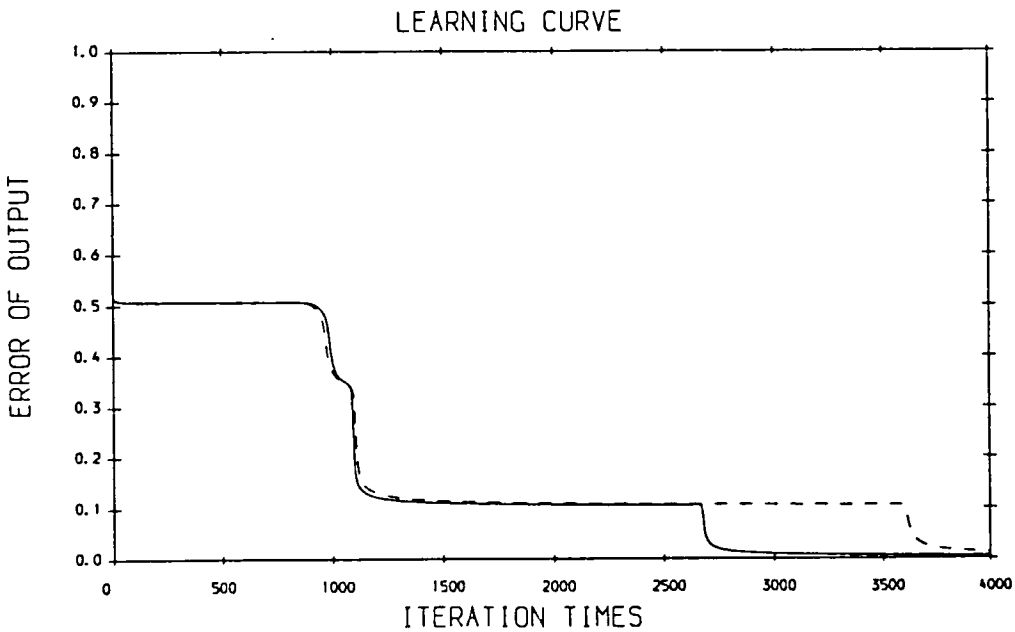


Fig-3.8

Learning curves for the 4-4-1 parity problem. Broken line stands for the learning curve of non-adaptive algorithm.

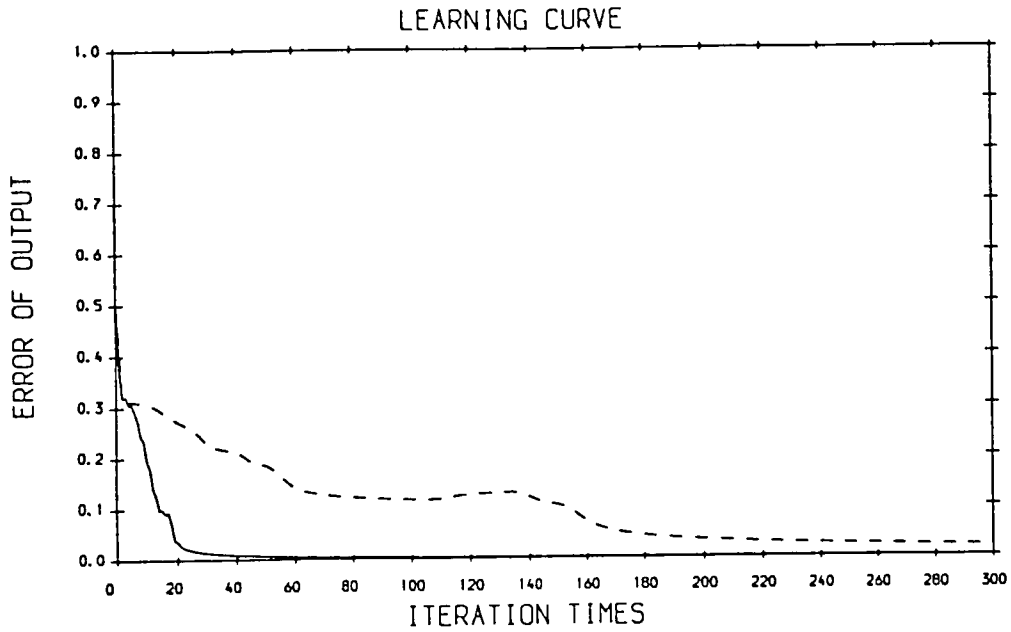


Fig-3.9

Learning curves of the 10-5-10 encoder problem. Solid line stands for the learning curve of the DSBP algorithm.

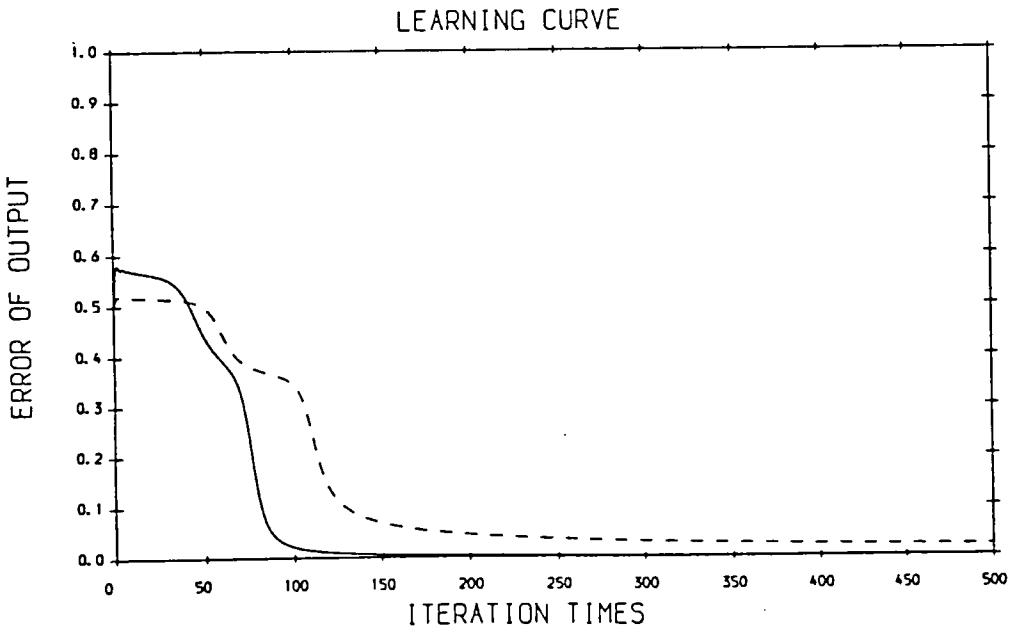


Fig-3.10

Learning curves of the XOR problem. Solid line stands for the learning curve of the DSBP algorithm.

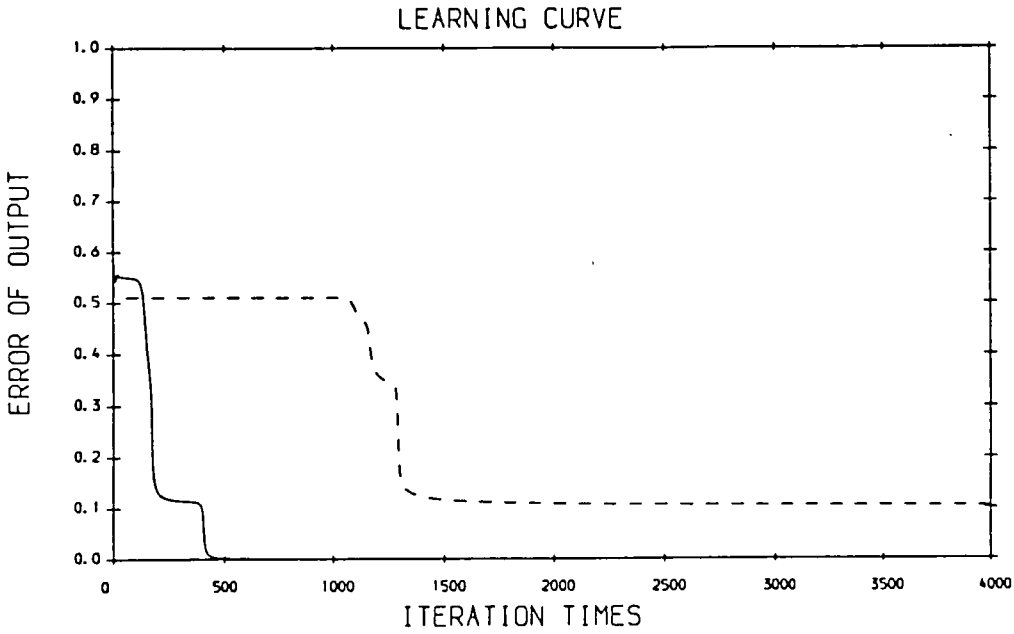


Fig-3.11

Learning curves of the 4-4-1 parity problem. Solid line stands for the learning curve of the DSBP algorithm.

Chapter Four
Generalization and Representation in MLP Networks

In the previous chapter, we have discussed the convergence of the Back-Propagation learning algorithm. However an efficient convergent learning algorithm can only guarantee the network will settle into a solution. As to whether there exists a solution, the representation problem, and how good the solution is, the generalization performance, are largely dependent on the structure of the network and the learning critera. In this chapter, we discuss the generalization problem of ANN especially MLP networks and the representation capability of a class of self-association networks.

4.1 Basic Problems of Generalization

The main objective of neural network research is to develop intelligent learning (or self-programming) machines, which are able to learn (or generalize) from examples. Generalization in this context is a kind of inductive inference. Although induction has been a central problem of philosophy, psychology and artificial intelligence for a long time, the formulation of a complete and comprehensive theory of induction is still the aim of current research [83].

For human beings, generalization ability is closely related to our species-specific heritage —— the capacity and the tendency to convert encounters with the particular into instances of the general[84], or in another words, from part to whole. The following simple example can demonstrate some basic features of generalization.

When the first three terms of a sequence are observed as 1, 2, 3, ... it is quite

natural to conjecture or generalize that the fourth term will be 4. However this may not be the case. If the generation mechanism is given by the recursive relation as

$$x_{n+1} = ax_n + b \tag{4-1}$$

Then from the first three terms, it is easy to calculate that $a = 1$ and $b = 1$. So the fourth term is actually 4. However if the generation mechanism is given by the recursive relation

$$x_{n+2} = x_{n+1} + x_n \tag{4-2}$$

then instead, the fourth term should be 5. As the generation mechanism can take many other forms, for example

$$x_{n+2} = ax_{n+1} + bx_n \tag{4-3}$$

or even higher order forms, the fourth term can take many other possible values. Selection of 4 as the fourth term is based on the simplest model of the observation. If it is a continuous observation process, and a human is trying to find out the underlying generation mechanism of the observed sequence, then when the actual observation does not match with the prediction of the model, the model will be adapted to be compatable with the observation. Thus generalization is a model building or rule extraction process. The generalization performance is judged on how well the extracted rule or model matches the observation.

In abstract form generalization can also be regarded as interpolation and extrapolation of data (or surface fitting) in abstract space [85] [86], each input-output pair in the learning sample set is a data point in the space. From this perspective, different models are just different surfaces. Thus for artificial neural networks, their potential models of the environment are heavily dependent on their structure and the representation strategy used to represent the input signal and output action.

This is especially true for networks which use the kind of learning algorithms which minimize the error function within the learning sample set, like Back-Propagation algorithm for example.

As the above simple example has demonstrated, learning problems are usually under-determined, or using a numerical analysis term ill-posed, so some kind of constraints must be imposed to restrict possible solutions. In the previous example, simplicity is the constraint. If generalization is regarded as surface fitting, then regularization theory can be used to impose necessary constraints on a learning problem. However the constraints derived from regularization theory, like minimal integrated curvature for example, may be considered too orientated to pure surface fitting rather than to the real world generalization [86], so a generalization theory based on high-level notions is proposed [86]. In this theory, some constraints imposed on generalization are geometrical properties like rotation, translation and scaling invariants. Although these geometrical constraints look closer to real world generalization, they also have their limitations. The geometrical invariant can help to obtain good generlization in some pattern recognition problems. However there are many other situations where geometrical invariancy is not valid. For example in a fractal pattern there are many variations of regularities, so the geometrical invariant in one sub-region may not be applicable to other regions. On the other hand, although minimal integated curvature may not sound relevent to high level concepts, it is a good constraint for some early vision problems. Actually, the constraints imposed are an expression of a priori knowledge of the problem. It is very difficult to judge absolutely which generlization constraint is the best and solution must be problem specific. Under situations where no a priori knowledge is available, some kind of statistical constraints have to be imposed for generalization [87] [88].

4.2 The Generalization of MLP Networks

In the previous section we have stressed that the generalization performance of ANN depends on their representation strategy and structure. In this section we consider how representation and structure can influence the generalization of MLP networks by considering some specific examples.

4.2.1 Representation and Multi-Solution Problems

Considered the MLP network used to implement the 4-2-4 encoding problem [19]. The network architecture is shown in Fig-4.1. In the 4-2-4 encoding problem, it is hoped that the network under training will find a transformation which can transform a redundant four-bit binary code into a suitable two-bit code to reduce the redundancy. In Table-4.1, Table-4.2 and Table-4.3, we have given three different simulation results on the 4-2-4 problem with different input codes. If we look at Table-4.1, we will find that after learning the internal representation or hidden unit states for corresponding input output pairs are quite close to 00, 11, 10 and 01 respectively. It seems that the network has found the transformation which can transform redundant four-bit binary codes into two-bit codes through generalization. However if we look at Table-4.2 and Table-4.3, we will find the internal representation in these situations are not so meaningful as in Table-4.1. Why did the Back-Propagation Algorithm fail to find a similar transformation in these two cases? Actually they are the same kind of problem, the only difference is the input representation. The answer is that because of the different input representation, the geometric constraints implied are different.

The four internal representations corresponding to the four inputs can be regarded as four points in a unit square on a two dimensional plane, and the weights

connecting the hidden units and the output units form four dividing lines on the plane. If two output codes have only one different bit, then there is one and only one dividing line between their internal state points. From this observation, it is quite easy for us to get the implied geometric constraints of the input and output patterns in Table-4.1, which can be shown as in Fig-4.2. The four lines L_1, L_2, L_3 and L_4 must form a four sides polyline, and four internal states A_1, A_2, A_3 and A_4 must lie in the region shown in Fig-4.2. As the Back-Propagation Algorithm has the tendency to move the points to the corners of the unit square or unit hypercubic, so A_1, A_2, A_3 and A_4 are very likely to be forced into the four corners of the unit square which correspond to the two-bit code shown in Table-4.1 internal states column. But for the input and output patterns in Table-4.2 and Table-4.3, the implied geometric constraints are different. They are shown in Fig-4.3 and Fig-4.4, Fig-4.3 for Table-4.2 and Fig-4.4 for Table-4.3. For the internal states pattern shown in Fig-4.3, A_1 stands for the internal state of output pattern 0000, it generally lies in the middle of A_2, A_3 and A_4. This is consistent with the results shown in Table-4.2. All these internal state patterns displayed in Fig-4.3 and Fig-4.4 are unlikely to be forced into corners of the unit square. This is why in Table-4.2 and Table-4.3 the Back-Propagation Algorithm did not find the internal states which are as meaningful as in Table-4.1.

Another problem associated with the generalization of MLP networks is the multi-solution feature of some learning tasks, the parity problem is an example. In [19] a highly organized structure has been discovered by an MLP network using a Back-Propagation algorithm which can solve the parity problem. However if we look at a simple version of the parity problem with only 3 bits, a 3-3-1 MLP network is used to implement it, and all the possible combinations of input patterns are used as learning samples, then there are three different ways of dividing input

space which can satisfy the condition set by the training sample set. These divisions are shown in Fig-4.5, Fig-4.6 and Fig-4.7. Which solution the Back-Propagation Algorithm adopts is dependent on the initial weights and learning parameters. In our simulations, we have obtained all three solutions for different simulations. Thus if the algorithm starts with random weights, which solution we will obtain is totally unpredictable, in other words, the generalization of the algorithm is totally unpredictable. This is the situation in which all possible input patterns are used as learning samples. For the cases where only a smaller learning sample set is available, the unpredictability could be even greater.

The generalization made by the MLP using the back propagation algorithm is based on the geometric constraints posed by the training sample set. Different training samples set up different constraints, but in some cases, for the same training samples, because of the multi-solution nature of the problem, (like the parity problem we mentioned above) there are many different ways of dividing the input space, all of which can satisfy the constraints. This will make the learning behaviour of the algorithm difficult to predict. Previous reference [18] stated '*The most obvious drawback of the learning procedure is that the error-surface may contain local minima so that gradient descent is not guaranteed to find a global minimium. However, experience with many tasks show that the network very rarely gets stuck in poor local minima that are significantly worse than the global minimum. We have only encountered this undesirable behaviour in networks that have just enough connections to perform the task. Adding a few more connections creates extra dimensions in weight space and these dimensions provide paths around the barriers that create poor local minima in the lower dimension subspace.*' From the above discusion, we will note that 'Adding a few more connections' will also increase the number of possible ways of dividing the input space while satisfying the constraints, and

thus the learning will be more unpredictable. This conclusion is also confirmed by the observation reported in [88]. In their experiment, the network is first set to a solution with $E = 0$. Then the weights are perturbed to another point in the weight space, and the network is re-trained. The new solution obtained by the network could be quite different from the original solution.

4.2.2 Initial Structure and Generalization

Fully connected MLP networks are the most commonly used artificial neural networks in research and application. The initial structure constraints imposed on them are very weak. Theoretically they can divide the input space into arbitrary shapes. It has been recently proved that any continuous mapping can be approximately represented by an MLP network[89]. This universal feature of MLP provides a high degree of flexibility, but also reduces its generalization capability and makes learning difficult. Even using a learning algorithm which can always converge to the global minima, in multi-solution cases the final solution reached may not be unique, and thus generalization is unpredictable. The small mean square error on the learning samples does not necessarily imply that generalization outside the learning sample set would be good. A universal MLP is more likely to store information into a memory structure than to identify the hidden structure of the process which generates the training samples[90]. Thus some prior structure (not so universal) may be necessary for nontrivial learning. It may be argued that a universal MLP can be regarded as a parameterized structure, for example, a weight of zero value means there is no connection between the two neurons associated with the weight. But the formidable high dimension of the parameter space and the complexty of the error function surface makes it impossible to guarantee the most suitable structure will be selected by the learning process. One may argue that an exhaustive search method can be used, but apart from the fact that it cannot solve

the multi-solution problem, it also makes the learning very slow as in some cases it becomes an NP problem[91]. Thus nontrivial self-programming in artificial neural networks can take place only if a priori knowledge about the enviroment in which the system is to learn is built into the system as an initial structure(Shepard in [92]).

The abstraction of neural networks with full connectivity and randomly distributed initial connection weights surely lacks biological plausibility. Real biological neural networks do have structure, and this structure determines how they process inputs. For example, the visual system has at least a dozen subsystems, each with elaborate internal and external structure[93][94]. These structures can only be inherited, because after the embryonic development phrase, organisms cannot develop any new connection between two neurons which are not initially connected.

Thus the initial structure is crucial to satisfactory generalization of artificial neural networks. The initial structure or internal constraints can guide the generalization process. We can illustrate this fact by a simple pattern recognition example. It is a two class pattern recognition problem. The sample points from class A are uniformly distributed in a circle or an ellipse. The sample points from class B are uniformly distributed around the circle or ellipse. The task for a recognizer is that after initial training when given new samples it should be able to classify them with a satisfactory precision. We used two different kinds of networks to solve this problem. One is a 2-10-1 fully connected MLP shown in Fig-4.8. The second network shown in Fig-4.9, we call a nonlinear perceptron, in which all weights between the input layer and hidden layer are fixed (they all have value of 1). The two hidden units on the left use a quadratic function as their activation function and the two hidden units on the right are just all-pass units (that is the output is equal to input). The weights between the hidden layer and the output layer are modifiable, the

output unit uses a sigmoid function as activation function. Both networks use the gradient descent learning algorithm (or back-propagation algorithm). Some simulation results are shown in Fig-4.10a, Fig-4.10b, Fig-4.11a and Fig-4.11b. Fig-4.10b and Fig-4.11b show the results of the 2-10-1 MLP, and Fig-4.10a and Fig-4.11a are for the nonlinear perceptron. The time for the MLP network to go through the training set is twice that for the nonlinear perceptron. The decision boundary or generalization of the nonlinear perceptron is better than that of the MLP. Actually the nonlinear perceptron is not as universal as the MLP. As the MLP can form piece-wise linear decision boundaries which can approximate any curve to any precision provided there are enough hidden units, the nonlinear perceptron can only form circular or elliptic decision boundaries. It is more problem specific. However it is this internal constraint that the nonlinear perceptron can only form circular and elliptic decision boundaries which provides better generalization in this specific enviroment. This is because it matches with the characteristic of the problem. Although the MLP is more universal, the back-propagation learning algorithm cannot make full use of hidden units, and some of them become redundant. As every hidden unit corresponds to a division line in Fig-4.10b and Fig-4.11b, it can be seen that some division lines have not been fully utilized to form decision boundaries.

Some other examples of initial structure facilitating learning and generalization can be found in[95], in which a hierarchical MLP network is designed for hand writing digits recognition, and the simulation results demonstrate some obvious improvement on generalization performance.

From the above discussion we can conclude that MLP networks using a Back-Propagation algorithm are a suitable choice for problems in which no a priori knowledge is available and the learning sample set is a typical representation of the functional space. However when some a priori information is available, as shown by

the above pattern recognition example, a custom-tailored network or learning algorithm which incorporates the a priori knowledge can always be advantageous. The representation strategy can also influence the generalization performance. It is suggested in [85] that for abstract learning problems it may be advantageous to attempt to code the data so as to produce a relationship which is as smooth as possible. However this may not be easy.

4.3 Hidden Units and the Representation Capability of Self-Association MLP

Although it has been proved that MLP networks can implement any nonlinear mapping if there are enough hidden units in the network, when the number of hidden units is bounded, their representation ability is also limited. In this section we consider the limitation on an MLP type self-association network used for feature extraction.

Feature extraction is an important technique for pattern recognition[96]. As the raw input information usually has high dimensionality and a lot of redundancy, compression techniques are used to form low dimensional feature vectors. This process is also called dimensionality reduction. One aspect of recent research is the application of neural networks to perform feature extraction or dimensionality reduction[97][98]. Ackley , Hinton and Sejnowski used the neural network for the compression coding problem[27], Baldi and Hornik analysed the linear unit neural network for information compression[99]. The neural network used for these applications are usually self-association (or auto-association) networks. There have also been attempts to use a self-association network to realize image compression [100].

A self-association neural network is shown in Fig-4.12. It is a fully connected feed-forward multi-layer network, with an input layer and an output layer , and one or more hidden layers. The input layer and the output layer have the same number of units. The task for this network is that after going through some training samples, when a pattern is presented to the input layer, the output layer should be able to reconstruct the same pattern. The hidden layer can be regarded as a feature detector, it compresses the input patterns into more concise form. Some researchers hope that this kind of network can be used to find useful transforms which can compress the input information into feature vectors[101]. For this network, an important question is that for a specific number of input units how many hidden units are needed to make the network capable of reconstructing every possible input pattern? The following theorem gives the answer to this question.

Theorem: For a self-association neural network like that shown in Fig-4.12, the number of input units is n, hidden units is p, usually $p \leq n$, all the input and output patterns are considered to be binary (black/white image). If perfect reconstruction is required, that is every possible input pattern should be able to be reconstructed on the output layer, then the necessary condition is p must satisfy $p \geq n - 1$. We assume that the activation function of hidden units is a sigmoid function (see Fig-1.2), and the activation function of the output units is step function (see Fig-1.3).

Proof: We denote the activation of hidden units as a vector **H** with dimension of p,

$$\mathbf{H} = [h_1, h_2, \ldots, h_p]^T \qquad (4-4)$$

each $h_k (k = 1, 2, \ldots, p)$ corresponds to an activation level of a hidden unit. The input to the output units can also be denoted as a vector **I** with dimension of n,

and the corresponding output as a n dimension **O**

$$\mathbf{I} = [i_1, i_2, \ldots, i_n]^T \tag{4-5}$$

each $i_k (k = 1, 2, \ldots, n)$ can be expressed as,

$$i_k = \sum_{j=1}^{p} w_{k,j} h_j + \theta_k \tag{4-6}$$

to facilitate the following discussion, we regard the $\theta_k (k = 1, 2, \ldots, n)$ as a weight which connects to an always active unit with active level of 1. Now if we expand the vector **H** by one row of 1, that is,

$$\mathbf{H} = [h_1, h_2, \ldots, h_p, 1]^T = [h_1, h_2, \ldots, h_{p+1}]^T \tag{4-7}$$

then (4-6) can be rewritten as;

$$i_k = \sum_{j=1}^{p+1} w_{k,j} h_j \tag{4-8}$$

now **H** is a p+1 dimension vector, the relation between **I** and **H** can be expressed as

$$\begin{bmatrix} i_1 \\ i_2 \\ \vdots \\ i_n \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,p+1} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,p+1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,p+1} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_{p+1} \end{bmatrix} \tag{4-9}$$

we denote the above matrix as **W**. The relation between the **I** and **O** can be expressed by a nonlinear operator **NF** as

$$\mathbf{O} = \mathbf{NF} \cdot \mathbf{I} \tag{4-10}$$

For any s-dimensional vector **X**, **NF** can be defined as,

$$\mathbf{NF} \cdot \mathbf{X} = [f(x_1), f(x_2), \ldots, f(x_s)]^T \tag{4-11}$$

f( ) is the step function defined by

$$f(x) = \begin{cases} 0, & \text{if } x \leq 0; \\ 1, & \text{otherwise.} \end{cases} \qquad (4-12)$$

So from (4-10) we can see that, for any i_k, if $i_k > 0$, then its corresponding o_k is 1; otherwise o_k is 0. If we regard every row of matrix **W** as a coordinate of a point in a (p+1)-dimensional space, then every output pattern corresponds to a division of these n points in a (p+1)-dimensional space into two groups by a linear hyperplane defined in (4-8). For any output unit which is 1, its corresponding point is above the hyperplane. otherwise it is below the hyperplane. Now we define F(m, p) as the number of all possible linear divisions of m points in a p-dimensional space (taking account of the direction of the hyperplane), we have $F(m,\ 1) = 2$ and $F(1,\ p) = 2$. If the distribution of the points satisfy the general position condition, then it can be shown that [102]

$$F(m+1,\ p) = F(m,\ p) + F(m,\ p-1)$$

and from the initial condition $F(m,\ 1) = 2$ and $F(1,\ p) = 2$, it can be proved that

$$F(m,\ p) = 2\sum_{k=0}^{p-1} C_{m-1}^{k} \qquad (4-13)$$

For an output layer with n units, the number of all possible output patterns is 2^n. So our original problem can be replaced by the problem of seeking the smallest p which satisfies $2^n \leq F(n, p+1)$. Now we will prove that p must be greater than or equal to n-1.

If $p = n-1$, then

$$\begin{aligned} F(n,\ p+1) &= F(n,\ n) \\ &= 2\sum_{k=0}^{n-1} C_{n-1}^{k} \\ &= 2 \times 2^{n-1} = 2^n \end{aligned} \qquad (4-14)$$

for any $p < n-1$, it is obvious that

$$F(n,\ p+1) = 2\sum_{k=0}^{p} C_{n-1}^{k} < 2\sum_{k=0}^{n-1} C_{n-1}^{k} = 2^n$$

So the smallest p which can guarantee $F(n,\ p) \geq 2^n$ is n-1. We have thus proved the theorem.

The condition $p \geq n-1$ is only a necessary condition, not a sufficient condition. So it is a lower bound. This is especially true since the last element of $\mathbf{H}\ h_{p+1}$ is fixed to 1, so it could limit the possible swap of polarity of hyperplanes. This can be demonstrated in a two dimensional example. Consider an auto-associative network with two output units and only one hidden unit. Then the ouputs can be described by

$$\begin{aligned} O_1 &= f_1(x) = f_1(w_1 h + \theta_1) \\ O_2 &= f_2(y) = f_2(w_2 h + \theta_2) \end{aligned} \qquad (4-15)$$

where $f_i(\ )$ is the step function defined in preceding paragraphs, w_1 and w_2 are the weights, θ_1 and θ_2 are the thresholds, h is the activation of hidden unit. If pair (x, y) is regarded as a point on a plane, and the point (x, y) for which $f_1(x) = 0$ and $f_2(y) = 0$ as the original point of the coordinate system, then the point which can produce 01 must lie in the second quadrant or on a positive half of the y-axis, the point which can produce 10 must lie in the fourth quadrant or on a positive half of the x-axis, and the point which can produce 11 must lie in the first qudrant. Thus if we want output pair (O_1, O_2) to be able to take all possible binary combinations as 00, 01, 10 and 11, then the distribution of four corresponding points (x, y) should be a pattern like that shown in Fig-4.13. As all possible (x, y) points lie on a straight line defined by

$$\begin{aligned} x &= w_1 h + \theta_1 \\ y &= w_2 h + \theta_2 \end{aligned} \qquad (4-16)$$

It is obvious that no straight line can go through all the four points displayed in Fig-4.13. That is to say a single hidden unit cannot reproduce all the binary combination patterns for two output units.

The above result seems quite pessimistic for the feed-forward self-association network, but it may not be so. If we demand only half of all possible input patterns are reconstructable, then the bound of p can be reduced by half. The proof is given as follows.

In this case the problem is to find the smallest p which satisfies

$$F(n,\ p) \geq 2^{n-1} \tag{4-17}$$

If n is an odd number, then let $n = 2s + 1$ (s is an integer). Because

$$\sum_{k=0}^{2s} C_{n-1}^{k} = 2^{n-1} \quad and \quad C_{n-1}^{k} = C_{n-1}^{n-1-k}$$

so we have

$$2\sum_{k=0}^{s} C_{n-1}^{k} > 2^{n-1}$$

$$2\sum_{k=0}^{s-1} C_{n-1}^{k} < 2^{n-1}$$

Thus to satisfy the condition (4-17), p must satisfy $p \geq [\frac{n-1}{2}]$. (where [ ] means truncate the decimal part) If n is an even number, it can be represented as $n = 2s + 2$. It is easy to obtain

$$2\sum_{k=0}^{s} C_{2s+1}^{k} = 2^{n-1}$$

so to keep (4-17) hold, p must satisfy $p \geq [\frac{n-1}{2}]$. We can conclude that for half reconstruction, p must satisfy

$$p \geq \left[\frac{n-1}{2}\right] \tag{4-18}$$

Finally we may conclude that the limitation on the number of hidden units has nothing to do with the learning algorithm, it is only related to the structure of the network. Thus it is a structure problem which cannot be solved by the learning algorithm. The lower bound of the number of hidden units cannot be circumvented by adding more hidden layers. From our foregoing analysis, it can be seen that the quantitative relation of the number of output units and that of the adjacent hidden layer for the perfect reconstruction condition is always held no matter how many hidden layers the network has. Thus it is a fundamental limitation.

The result provided by the theorem seems quite pessimistic, although it may not be so. It is only under the perfect reconstruction condition that we would have to have the same number of hidden units as that of the input. But in most real application problems, useful patterns are only a small portion of all possible patterns, many of them are meaningless for a specific problem. In these cases we can use less hidden units than that demanded for perfect reconstruction. Actually, we have already shown that if only half of all possible patterns are needed, the bound of p can also be reduced by half.

In this and the previous chapters we have discussed some properties of ANN, especially MLP networks. In the following chapters we will discuss potential applications of ANN.

Input Pattern	Internal Representation	Output Pattern
0 0 0 1	0.0 0.0	0 0 0 1
0 0 1 0	1.0 1.0	0 0 1 0
0 1 0 0	1.0 0.0	0 1 0 0
1 0 0 0	0.0 1.0	1 0 0 0

Table-4.1

Input Pattern	Internal Representation	Output Pattern
0 0 0 0	0.6 0.4	0 0 0 0
0 0 0 1	0.0 0.0	0 0 0 1
0 0 1 0	1.0 0.0	0 0 1 0
0 1 0 0	0.3 1.0	0 1 0 0

Table-4.2

Input Pattern	Internal Representation	Output Pattern
0 0 0 1	1.0 0.0	0 0 0 1
0 0 1 1	1.0 0.6	0 0 1 1
0 1 1 1	0.7 1.0	0 1 1 1
1 1 1 1	0.0 1.0	1 1 1 1

Table-4.3

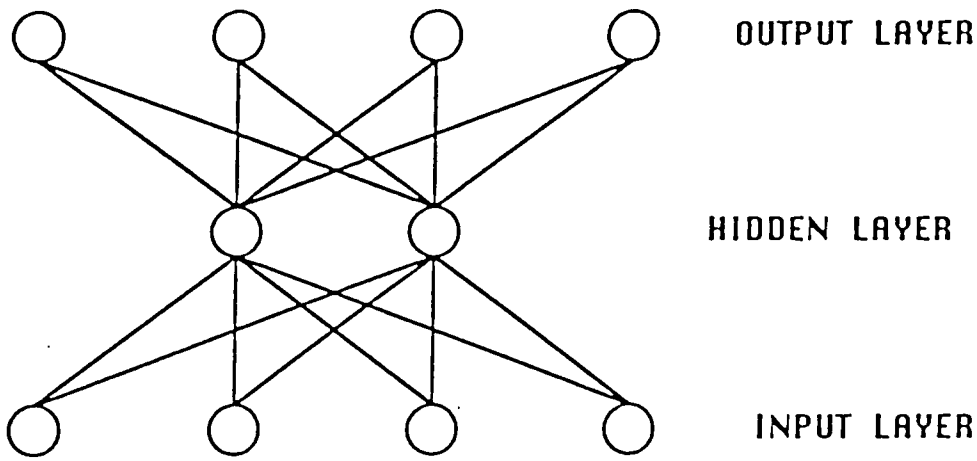


Fig-4.1 MLP network for 4-2-4 encoding problem

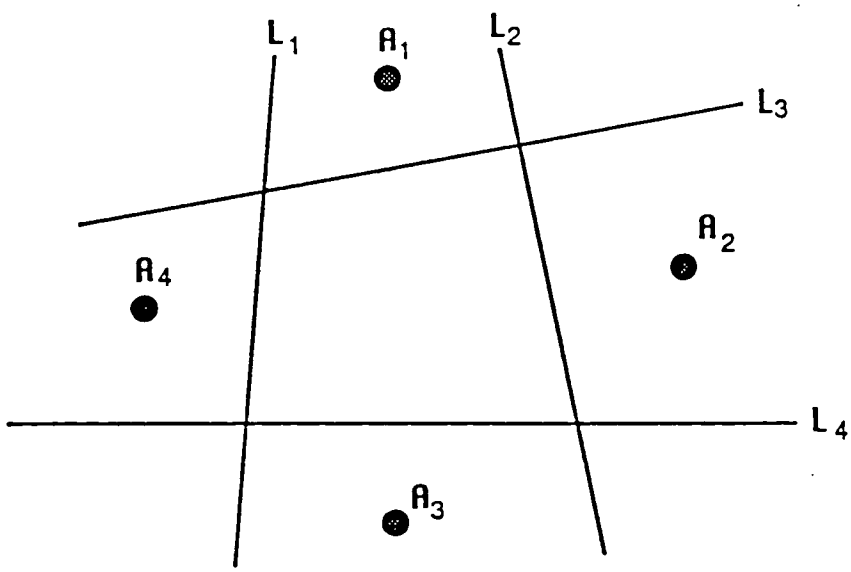


Fig-4.2

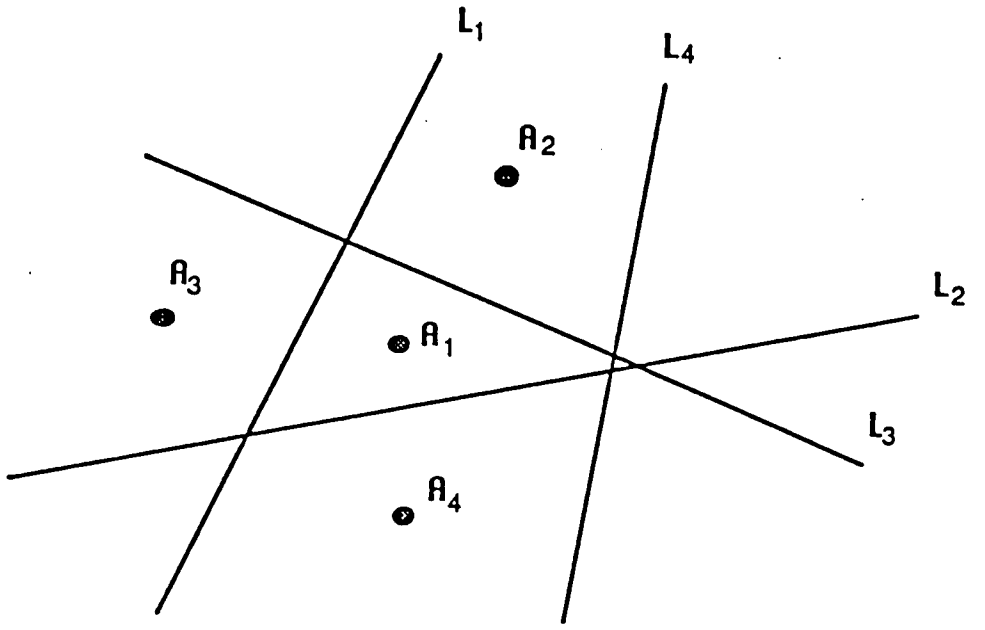


Fig-4.3

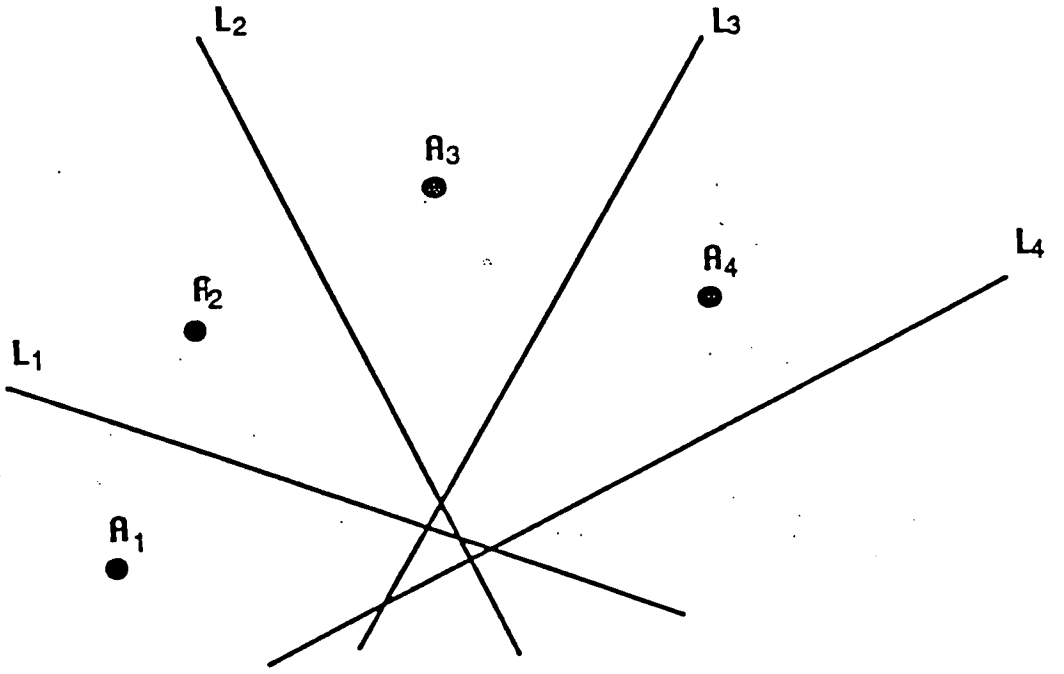


Fig-4.4

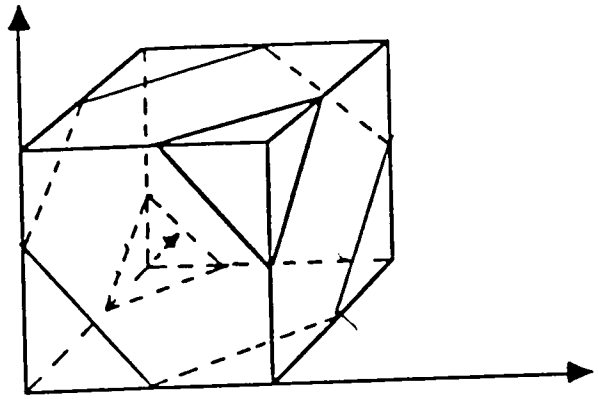


Fig-4.5

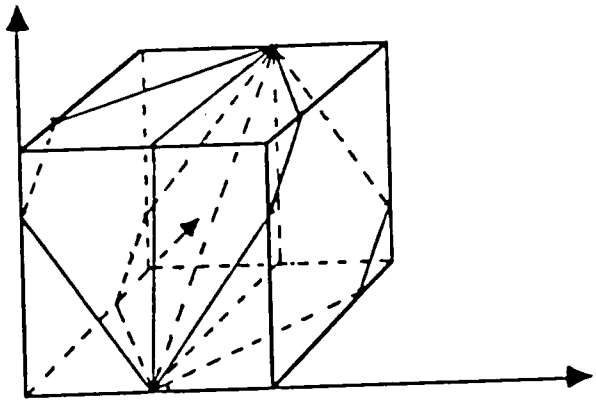


Fig-4.6

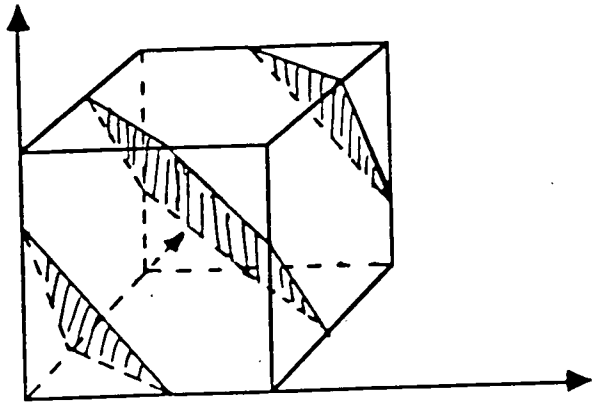


Fig-4.7

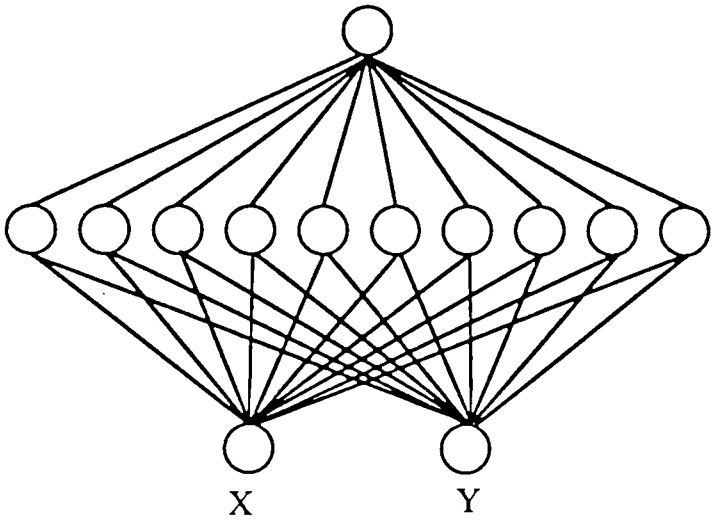


Fig-4.8 2-10-1 MLP Network

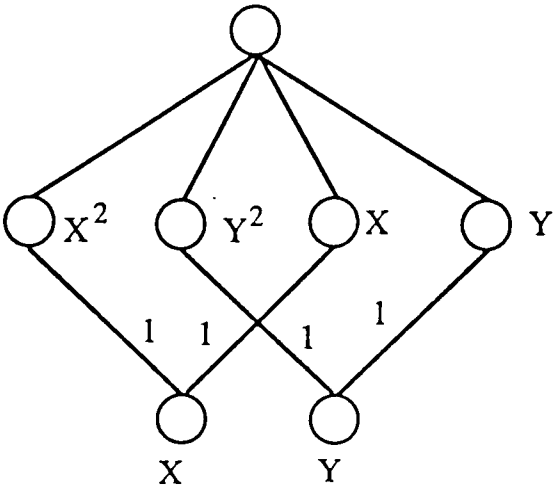


Fig-4.9 The nonlinear perceptron

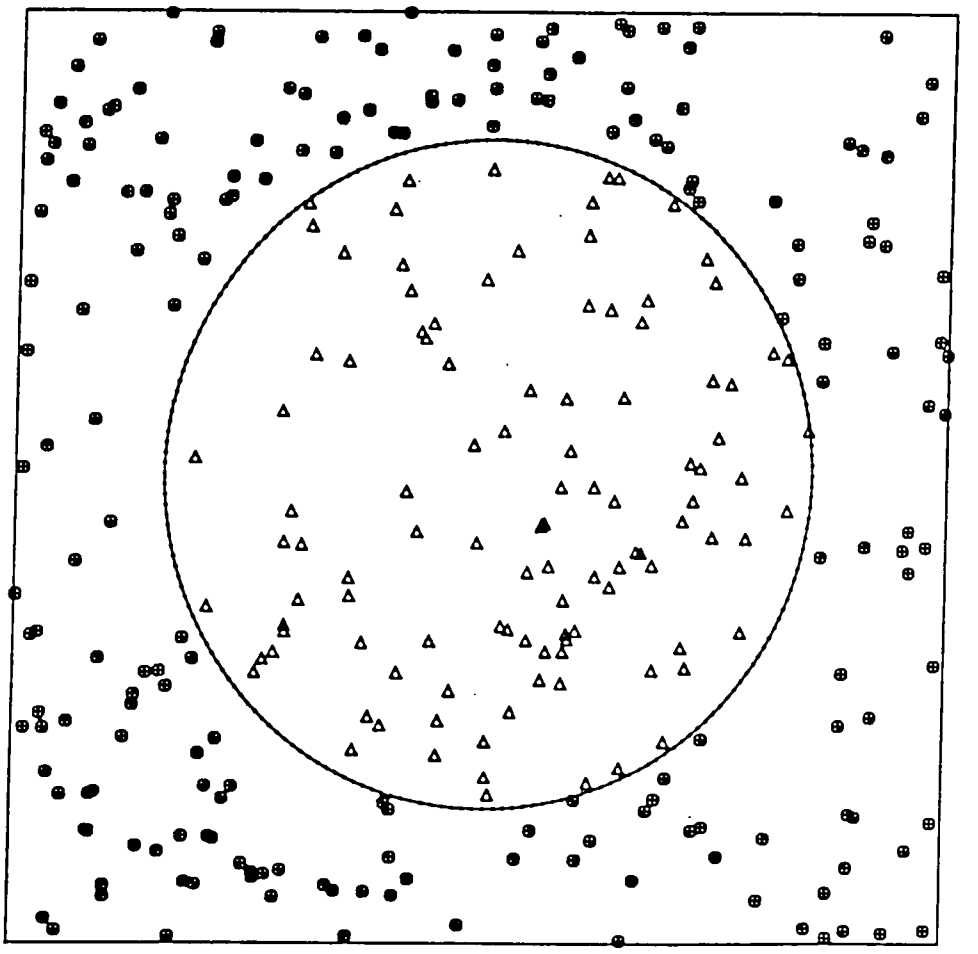


Fig-4.10a

This figure shows the distribution of training samples and the decision boundary learned by the nonlinear perceptron.

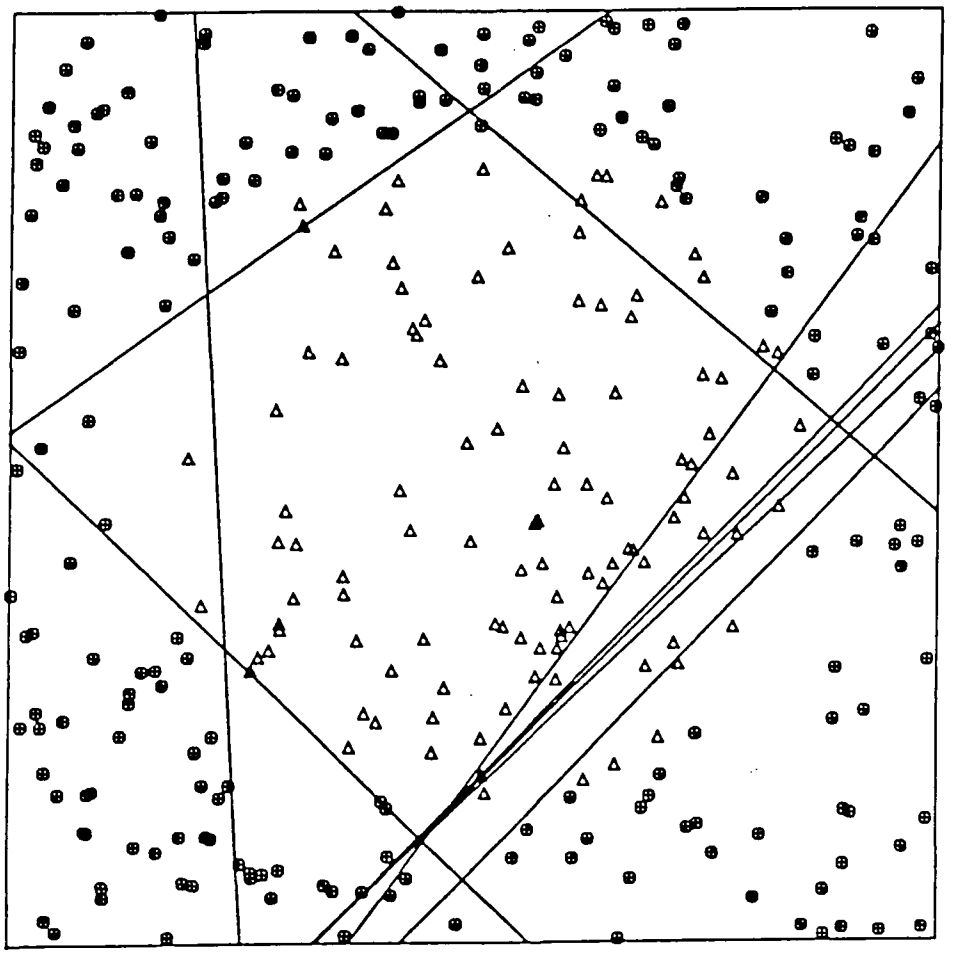


Fig-4.10b

This figure shows the distribution of training samples and the decision boundary learned by the 2-10-1 MLP. THe distribution is the same as Fig-4.10a

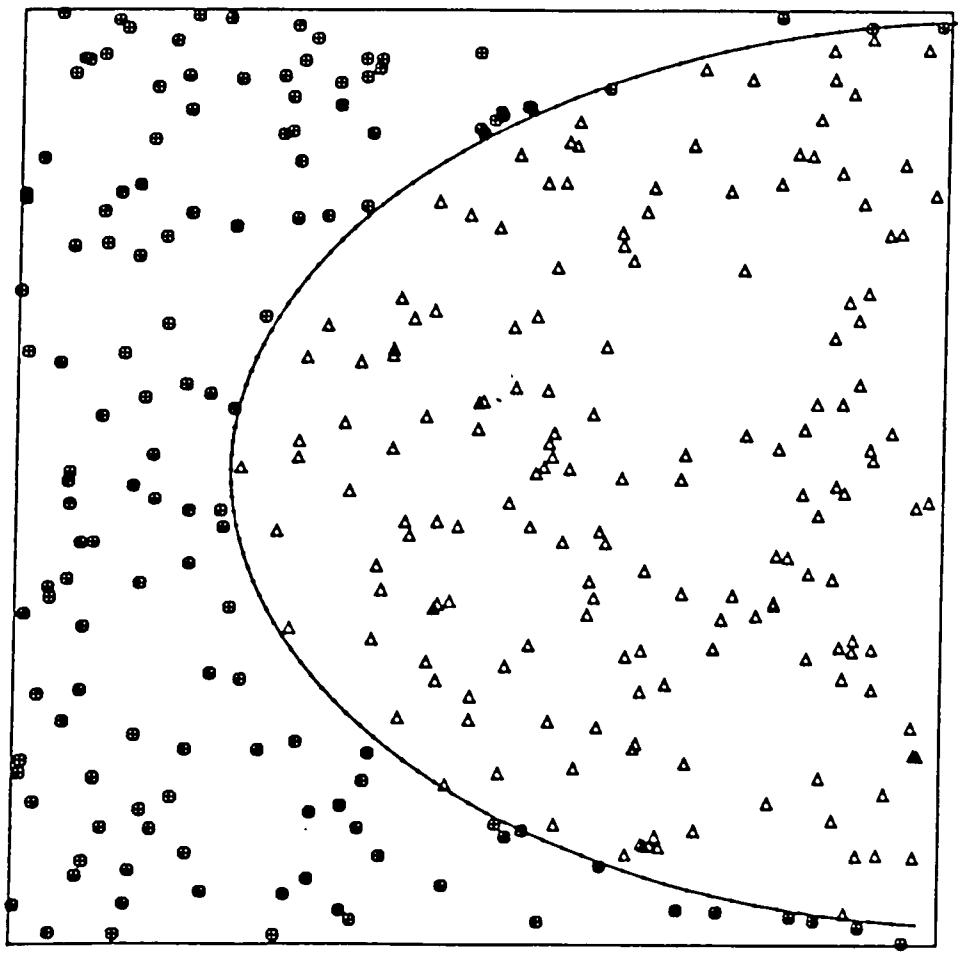


Fig-4.11a

This figure shows the distribution of training samples and the decision boundary learned by the nonlinear perceptron.

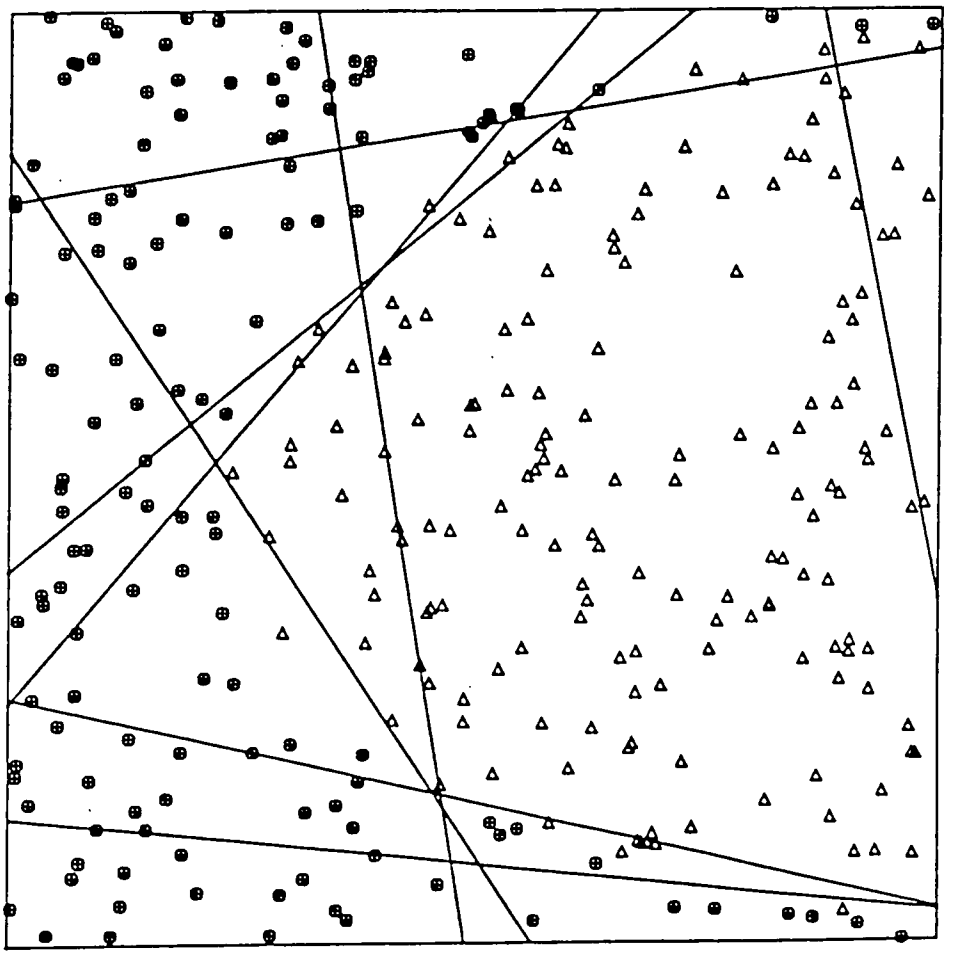


Fig-4.11b

This figure shows the distribution of training samples and the decision boundary learned by the 2-10-1 MLP. THe distribution is the same as Fig-4.11a

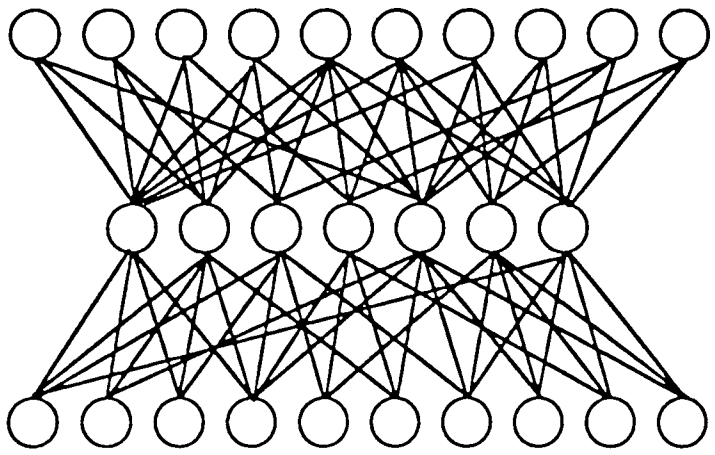


Fig-4.12 Feed-forward self-association network

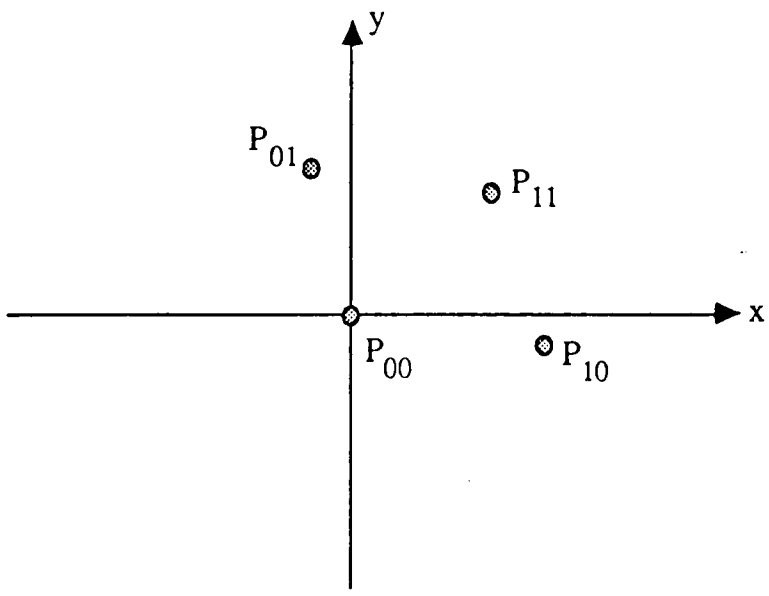


Fig-4.13 Output vector distribution pattern

Chapter Five
MLP Networks for Nonlinear System Identification

5.1 Introduction

Representation and identification are fundamental problems in system theory and signal processing. One way of establishing a mathematical model of a given system is by analyzing the physical mechanisims governing the operation of the system, and then write down the differential or difference equations which describe the operation of the system based on physical laws. This approach may not be possible in many situations, because of our incomplete knowledge of the system. An alternative approach is to build a system model based on observation of the input and output of the system. Thus the representation and identification of systems with given input-output relationship is an important problem for system research. For linear time-invariant systems this problem has been well studied and many methods and algorithms are available [103] [104]. However for nonlinear system identification the problem is much more complex and difficult.

One way to describe nonlinear systems is to use the Volterra series[105]. For a system with output time function y(t) and input excitation x(t), the input and the output relations can be expressed in the form

$$\begin{aligned} y(t) = & \int_0^\infty h_1(\tau_1)x(t-\tau_1)d\tau_1 + \int_0^\infty \int_0^\infty h_2(\tau_1,\tau_2)x(t-\tau_1)x(t-\tau_2)d\tau_1 d\tau_2 \\ & + \int_0^\infty \int_0^\infty \int_0^\infty h_3(\tau_1,\tau_2,\tau_3)x(t-\tau_1)x(t-\tau_2)x(t-\tau_3)d\tau_1 d\tau_2 d\tau_3 \\ & \cdots + \int_0^\infty \cdots \int_0^\infty h_n(\tau_1,\tau_2,\ldots,\tau_n)x(t-\tau_1)x(t-\tau_2)\cdots \\ & \cdots x(t-\tau_n)d\tau_1 \cdots d\tau_n + \cdots \end{aligned}$$

This series is called a Volterra series, and the functions $h_n(\tau_1,\ldots,\tau_n)$ are called

the Volterra kernels of the system. The analysis assumes that the system is time invariant. However there are two basic difficulties associated with the practical application of the Volterra series. The first difficulty concerns the measurement of the Volterra kernels of a given system and the second concerns the convergence of the series. Other functional series expansion methods for nonlinear system representation include Wiener series [105] and the Uryson operator [106]. In spite of the theoretical promise, all these models have some practical difficulties for general applicability. While the input-output finite order differential or difference equation model achieves wide acceptance in representation and identification of linear systems, it is natural to try to extend the input-output model to nonlinear systems. The input-output difference equation model for discrete nonlinear systems was proposed by Leontaritis and Billings in [107]. Recently with the development of research in ANN, Narendra and Parthasarathy proposed a nonlinear system identification scheme based on an finite order input-output difference equation model and MLP network [108]. There are many open questions concerning the theoretical and practical issues of the identification of nonlinear systems with neural networks. Examples are the excitation condition and the convergence of the weights. In this chapter, we discuss some of these fundamental problems and provide some computer simulations. Because of the theoretical difficulties of nonlinear systems, computer simulation is still an indispensible approach for nonlinear system study.

5.2 Identification of Nonlinear Systems with Static Nonlinearity

Many nonlinear systems can be described by the recursive difference equation

$$x(n+1) = f(x(n), x(n-1), \ldots, x(n-p+1), u(n), u(n-1), \ldots, u(n-q+1)) \quad (5-1)$$

where $x(i)$ is the output of the system, and $u(i)$ is the input to the system. It can

be proved that under some mild conditions nonlinear systems which operate near the equilibrium point can always be described by a difference equation of the form given in (5-1) [107]. From this expression we can see the essence of applying the neural network to nonlinear system identification is to use the neural network to approximate the nonlinear mapping f( ) in (5-1). Theoretically, the MLP neural network can approximate any continuous nonlinear mapping to any precision, provide there are enough hidden units [89]. However in a practical implementation, how well a MLP network approximates a given nonlinear function depends on many factors, such as number of learning samples and network structure. As the foundation of this nonlinear system identification scheme is a static nonlinear mapping approximation, we initially discuss the identification of static nonlinear mappings.

5.2.1 Static Nonlinear Mappings

The identification of a static nonlinear mapping can be implemented with the structure shown in Fig-5.1, where the back propagation algorithm is used to adjust the weights of the neural network. Fig-5.2 shows the simulation results of using a 1-20-10-1 neural network to approximate the nonlinear function

$$f(x) = x^3 + 0.3x^2 - 0.4x$$

and Fig-5.3 shows the simulation results of using a 1-20-10-1 neural network to fit the nonlinear function

$$f(x) = \frac{1}{4}(x^4 - 2.94x^2 - 0.44x - 0.5)$$

It may be observed that within the learning section, the fitting of the curves is almost perfect. However if we expand the displayed sections to [-2, 2], as shown in Fig-5.4 and Fig-5.5, we find the fit outside the learning section is disappointing. This poor generalization is the intrinsic weakness of unstructured neural networks.

Fig-5.6 shows the simulation result for fitting the nonlinear function

$$f(x) = 0.8sin(\pi x) + 0.2sin(5\pi x)$$

and Fig-5.7 is for

$$f(x) = -0.31x + 0.8sin(\frac{\pi x}{2}) - 0.1sin(10\pi x)$$

The results are not as good as those in Fig-5.2 and Fig-5.3. From our own simulation and that in Narendra's paper[108], we suggest that the more singular points (the points where the derivative of the function is zero) the function has, the more difficult it is to fit with a neural network. That is more hidden units and training time are needed.

As it is usually difficult to envisage the shape of multi-variable functions, we define a Discrepency Estimation Function (DEF) d(x) in our simulation,

$$d(x) = \max_{\mathbf{Y} \in S_x} |f(\mathbf{Y}) - NN(\mathbf{Y})|$$

where $\mathbf{Y}$ is the variable vector, and S_x is a shell defined as $S_x = \{\mathbf{Y} \mid \mathbf{Y}^T\mathbf{Y} = x^2 \quad or \quad \|\mathbf{Y}\| = x\}$ So d(x) can be used to measure the discrepency between the nonlinear function and the neural network. To save computation time, we used the random samples in S_x to estimate the d(x). The number of samples is 2^{dim}, dim is the dimension of the $\mathbf{Y}$. For example, for a three variable function, eight random samples are used for every fixed x to estimate the d(x). If eight random samples are denoted as $\mathbf{Y}_1, \mathbf{Y}_2, \ldots, \mathbf{Y}_8$, and $\|\mathbf{Y}_i\| = x$ $(i = 1, 2, \ldots, 8)$, then

$$d(x) \approx \max_{i \in \{1,2,\ldots,8\}} |f(\mathbf{Y}_i) - NN(\mathbf{Y}_i)|$$

Fig-5.8 shows the d(x) for the fitting of

$$f(x_1, x_2) = \frac{x_1 x_2 (x_1 + 2.5)(x_1 - 1.0)}{1 + x_1^2 + x_2^2}$$

with a 2-20-10-1 neural network. Fig-5.9 and Fig-5.10 show the surface of $f(x_1, x_2)$ and the neural network respectively. In Fig-5.11, the error surface $e(x_1, x_2)$ is displayed, it is defined as

$$e(x_1, x_2) = |f(x_1, x_2) - NN(x_1, x_2)|$$

We can see that the central region is low and relatively flat, it corresponds to the initial part of d(x) in Fig-5.8. The general tendency of $e(x_1, x_2)$ and the d(x) is the same. This can also be shown in Fig-5.12 and Fig-5.13, in which

$$f(x_1, x_2) = (x_1 - 0.9)(x_2 + 0.3)$$

This gives us confidence to use d(x) as a coarse estimation of fitting error between a nonlinear function f( ) and the neural network.

5.2.2 Dynamic Systems with only Static Nonlinearity

Now we consider the identification of nonlinear systems with only a static nonlinearity (in [108] they are called Model-I nonlinear systems) with MLP networks. A second order Model-I nonlinear system can be described by the difference equation of the form

$$x(k+1) = a_1 x(k) + a_2 x(k-1) + f(u(k)) \qquad (5-2)$$

which is a linear dynamic system with a nonlinear input mapping, u(k) as the exciting signal. We used a 1-20-10-1 neural network to identify the nonlinear function f( ) which has the form

$$f(u) = u^3 + 0.3u^2 - 0.4u$$

The neural network has two hidden layers, the first hidden layer has 20 units and the second has 10 units. The difference equation of the neural system is

$$\hat{x}(k+1) = a_1 \hat{x}(k) + a_2 \hat{x}(k-1) + NN(u(k)) \qquad (5-3)$$

NN( ) represents the neural network. If the error function e(k) is defined as $e(k) = x(k) - \hat{x}(k)$, then from equation (5-2) and (5-3) we have

$$e(k+1) = a_1 e(k) + a_2 e(k-1) + (f(u(k)) - NN(u(k))) \qquad (5-4)$$

The partial derivative of e(k+1) with respect of w_{ij} (w_{ij} is a weight of the neural network) is

$$\frac{\partial e(k+1)}{\partial w_{ij}} = a_1 \frac{\partial e(k)}{\partial w_{ij}} + a_2 \frac{\partial e(k-1)}{\partial w_{ij}} - \frac{\partial N(u(k))}{\partial w_{ij}} \qquad (5-5)$$

Equation (5-5) describes a linear system which has the same form as the linear part of equation (5-2) and thus is known. The input term $\frac{\partial N(u(k))}{\partial w_{ij}}$ can be calculated by the back propagation algorithm. As the partial derivative is often used as a measurement of sensitivity, the structure used for computing the partial derivative, like the back propagation algorithm, is called a sensitivity network[108]. Thus the structure for the identification of a Model-I nonlinear system is shown in Fig-5.14. Strictly speaking, the equation (5-5) is only valid when the weights of the neural network are constant. As the weights are always changing in the identification process, the partial derivative obtained from equation (5-5) is only an approximate estimation. Thus, unlike the back propagation algorithm, the algorithm used here is not a strict gradient descent algorithm. The simulation results are shown in the following figures. (Fig-5.15—5.17)

It seems the transient time of the back propagation is very short and it can trace the output of the plant very quickly. But actually this is not completely true. If we stop the weight updating of the neural network, the output of the neural system will fail to trace the output of the plant. This can be seen very clearly in Fig-5.15, in this case the weight updating stopped at time 300. The reason for this is that in the identification algorithm, we have two dynamic processes, one is described by the difference equation (5-3), and the other is the weight updating process. During

the period before time 300, it is the weight updating process itself that is tracing the dynamic process, rather than the process defined by the equation (5-3).

To provide more evidence for our argument, we repeated the above simulation with another two different excitations. One is a triangular wave, which can be expressed as

$$u(k) = \begin{cases} 0.01(t - 2k \times 100), & \text{if } 2k \times 100 \leq t < (2k+1) \times 100; \\ 0.01(2k \times 100 - t), & \text{if } (2k-1) \times 100 \leq t < 2k \times 100. \end{cases}$$

where k is an integer, the results are shown in Fig-5.16. The results of random excitation are shown in Fig-5.17. It is quite clear that in Fig-5.16, the back propagation algorithm can trace the output of the nonlinear system quite well after only 50 iterations, just as in Fig-5.15. But Fig-5.17 shows the results obtained with a uniform random excitation. In this case even after time 350 the tracing is still very poor. This provides strong evidence that it is not the fast and correct identification of the system which permits the output of the neural system to trace the output of the nonlinear system, but the weight updating process itself that is tracing the nonlinear system. The waveforms in Fig-5.15 and Fig-5.16 are slowly changing and regular, so it is possible for the weight updating process to trace them. Actually we see that the during the training period, the output of the neural system is almost a slightly delayed replica of the output of the nonlinear system. However for the random excitation case, the output of the nonlinear system is changing so dramatically and irregularly that only when the neural network has approximated the nonlinear function to a specific precision can the neural system trace the nonlinear system. The randomness needed here is to force the identification process into action. Otherwise the tracing of the output of the nonlinear system is realised by the continuous changing of the weights. It is more like a weight convergence condition and it is different from the persistent excitation in adaptive system theory [109] [110]. Persistent excitation ensures that the excitation should be rich enough to

make every aspect of the system identifiable. But the randomness needed here is employed in a different sense. Actually we are only going to identify the nonlinear function f(u) for $u \in [-1, 1]$, the excitation $u(k) = sin(\frac{2\pi k}{250})$ is revealing enough. Fig-5.18a and Fig-5.18b show the simulation results after 199700 learning iterations. Fig-5.18a shows the output traces and Fig-5.18b shows the curve of the nonlinear function f(u) in equation (5-2) and that of the 1-20-10-1 neural network. Although the learning is extremely long (199700!), the identification shown in Fig-5.18b is poor and tracing broke down after the learning (weight updating) stopped (see Fig-5.18a). On the other hand the tracing during the learning period is perfect. It provides a strong case that the irregular excitation should be used to break down the tracing by the weight updating, and force the identification process into action.

Randomness can actually lead to better identification. Fig-5.19a and Fig-5.19b show the simulation results for random excitation. The excitation is an independent random process with a uniform distribution over [0, 1]. We can see from Fig-5.19b that the identification of f(u) for $u \in [0, 1]$ is perfect. We can also use a sinusoidal excitation to get similar results. If the sinusoidal excitation has the form $u(k) = sin(\frac{2\pi k}{a})$, and a is irrational number, the trace of u(k) will appear irregular. As it is impossible to implement an irrational number on a digital computer, we used the function $u(k) = sin(\frac{2\pi k}{0.876513})$ instead. The identification results are shown in Fig-5.20. Thus, what is necessary for correct identification is some irregularity in the excitation.

As it has plagued all the neural network applications, it is not surprising that generalization is also a problem in nonlinear system identification. If we extend the display section of the curve of neural network in Fig-5.19b, as is shown in Fig-5.21, then we will find that the fitting outside the section [0, 1] is much worse than within [0, 1]. So for a real application, the magnitude of the excitation should be sufficient

to cover the whole dynamic range of interest.

To study the noise immunity of the back propagation algorithm for model-1 system identification, we added some random noise with normal distribution to the system, as shown in Fig-5.22. The simulation results are shown in Fig-5.23. Using the noise with the standard deviation of 0.333 (or variance of 0.1), the identification is reasonable, but obviously worse than that shown in Fig-5.20. If we increase the standard deviation to 0.5, we can see in Fig-5.24 that the identification is very poor. If the noise is added at the input port of the neural system as shown in Fig-5.25, rather than at the output of the nonlinear system, the identification will also be unsatisfactory. This can be seen in Fig-5.26, where the noise level is the same as in Fig-5.23, but the identification is inferior.

5.3 Identification of Systems with Nonlinear Dynamics

In this section we discuss the identification of nonlinear systems which have nonlinear dynamics but linear input excitation. They are called Model-II nonlinear systems in [108]. The system can be described by a nonlinear difference equation as

$$x(k+1) = f(x(k), x(k-1), \ldots, x(k-n+1)) + \sum_{i=0}^{m-1} b_i u(k-i) \qquad (5-6)$$

where the coefficients b_i are known, and the f( ) is an unknown continuous function. To simplify the simulation and the analysis, (5-6) can be replaced by the following equation

$$x(k+1) = f(x(k), x(k-1), \ldots, x(k-n+1)) + u(k) \qquad (5-7)$$

Because all the coefficients b_i and the excitation u(k) are known, there is not much difference between using equation (5-6) or (5-7) for simulation.

In this identification problem, the neural system which is used to model the plant can be represented as

$$\hat{x}(k+1) = NN(\hat{x}(k), \hat{x}(k-1), \ldots, \hat{x}(k-n+1)) + u(k) \qquad (5-8)$$

where $\hat{x}(k)$ is the estimation of x(k), and u(k) is the known excitation which is the same as that in (5-7). From equation (5-7) and (5-8), the discrepancy between the plant and the neural system can be calculated as

$$\begin{aligned} e(k+1) &= x(k+1) - \hat{x}(k+1) \\ &= f(x(k), \ldots, x(k-n+1)) - NN(\hat{x}(k), \ldots, \hat{x}(k-n+1)) \quad (5-9) \end{aligned}$$

the e(k) is used in the identification processes to adjust the neural network to minimize the discrepency between the plant and the neural network. As the parallel identification scheme described in (5-9) is difficult to converge even in linear identification, in our following study the series model is used. The architecture of identification is shown in Fig-5.27, and equation (5-9) can be replaced by the equation

$$e(k+1) = f(x(k), \ldots, x(k-n+1)) - NN(x(k), \ldots, x(k-n+1)) \qquad (5-10)$$

From equation (5-10) we can see that the identification problem in this case is almost the same as the function fitting problem which is discussed in section 2. The difference here is that the samples used for calculating e(k) are determined by the property of the system to be identified. However in the function fitting case the samples can be selected arbitrarily. Thus to obtain a satisfactory identification, the system and the excitation should meet some demand .

5.3.1 System Dynamic Properties and Identification Performance

First we consider the linear system situation, in this case the function f( ) has the form

$$f(x(k), \ldots, x(k-n+1)) = a_1 x(k) + \cdots + a_n x(k-n+1) \qquad (5-11)$$

If a FIR adaptive filter and LMS algorithm are used for the identification, it can be shown that under the persistent excitation condition, the coefficients of the filter will converge to a_i exponentially. This also means if a single layer linear perceptron is used as NN( ), it will converge to the linear function f( ) under a persistent excitation condition. However if a Multi Layer Perceptron is used, as is the case in this chapter, the persistent excitation can no longer guarantee the convergence of NN( ) to the f( ). In the adaptive filter case, the structure of the filter is the same as that of the plant, so identification is essentially parameter estimation and the convergence of the parameters under the persistent excitation implies the convergence of NN( ) to the f( ). In the case of a MLP neural network, its structure is different from the plant and it is more universal than a single layer perceptron. This universality gives it a powerful representation ability, but also renders poor generalization (see Chapter 4). The generalization mechanism of the MLP neural network is the interpolation between the learning samples and extrapolation outside the learning region. To make the interpolation match the function which produces the learning samples, the learning samples should be very dense within the learning region of the input space, and generally the extrapolation outside the learning region is very poor, as shown previously in this chapter. Thus to obtain satisfactory identification, the learning samples should cover the whole input domain which one is interested in, and have sufficient density. The persistent excitation cannot guarantee this, only a more general excitation and the system with specific properties can ensure identification. Because the learning samples are actually the states of the system, they lie on

the phase trace which is determined by the excitation and the system properties. Thus the distribution of learning samples is closely related to the excitation and the system properties.

There are several system properties which can influence the phase trace. First, we consider the controllability of the system. In system theory, controllability means any system state can be reached within finite time with an appropriate excitation[111]. Therefore if a system is controllable, theoretically the phase trace can densely cover the whole phase space under an appropriate excitation. (In some circumstances this is random excitation.) This is precisely the requirement in using a MLP neural network for system identification.

For a linear system which is described by the state equation

$$\vec{X}_{k+1} = \mathbf{A}\vec{X}_k + \vec{B}u_k \qquad (5-12)$$

where $\mathbf{A}$ is the transfer matrix, $\vec{X}_k$ is the state vector, the condition of controllability is that the $\mathbf{C}$ matrix defined by

$$\mathbf{C} = [\vec{B}, \mathbf{A}\vec{B}, \ldots, \mathbf{A}^{n-1}\vec{B}]$$

has full rank. For the linear system represented by the equation (5-11), the $\mathbf{A}$ matrix is

$$\mathbf{A} = \begin{pmatrix} a_1 & a_2 & \ldots & a_{n-1} & a_n \\ 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & 0 \end{pmatrix}$$

and $\vec{B} = [1, 0, \ldots, 0]^T$. It is easy to verify that the matrix $\mathbf{C}$ has the form

$$\mathbf{C} = \begin{pmatrix} 1 & \times & \times & \ldots & \times \\ 0 & 1 & \times & \ldots & \times \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \end{pmatrix}$$

irrespective of the values of the $\times$ elements , the C matrix is absolutely full rank. So the linear systems described by the equation (5-11) are always controllable.

Besides the controllability, the phase trace is also influenced by the correlation property or bandwidth of the system. Although the controllability guarantees that every corner of the state space is reachable under appropriate excitation, the distribution of the phase trace is more influenced by the bandwidth of the system if the excitation is not specifically designed. Under white noise excitation, the output of a narrow band system will be a highly correlated process, the phase trace will be more likely restricted in a narrow region along the diagonal line, although there is the possibility that the phase trace will reach every corner of the state space ultimately. The distribution of phase trace is highly non-uniform in this case. To obtain a satisfactory identification in a large portion of state space, the system must be wideband or the excitation should be specifically designed. It is only under these conditions, that the coverage of the state space by the learning samples will be dense and complete, assuming the learning time is of sufficient length.

In Fig-5.28 to Fig-5.32, the simulation results for a narrow band system are shown. In this case the plant to be identified is a linear system which can be described by the difference equation

$$x(k+1) = 1.6x(k) - 0.65x(k-1) + u(k) \qquad (5-13)$$

This is a narrow band system, and the output is highly correlated even under random excitation. Its phase portrait is shown in Fig-5.28, and it is clear that the phase trace is only concentrated in the diagonal region, although the excitation is random. Thus the learning samples for the neural network are also concentrated in the diagonal region. After 99,900 learning iterations, the error surface and the curve of d(x) are shown in Fig-5.29 and Fig-5.30. From Fig-5.29, it is obvious

that the discrepency between the neural network and the plant is small only in the diagonal region. This means the generalization will be poor outside this region of state space. Fig-5.31 shows the tracing performance of the neural system under the learning random excitation. The learning stopped at iteration 99,900, but the tracing is still quite good even after that time. If the excitation is replaced by a square wave which has the form

$$u(k) = \begin{cases} -0.04, & \text{if } 100 \times n \leq k \leq 100 \times n + 50; \\ 0.04, & \text{otherwise.} \end{cases}$$

where n is a integer, the tracing performance is shown in Fig-5.32. The generalization is not perfect.

If the equation (5-13) be changed to the form

$$x(k+1) = 0.5x(k) - 0.9x(k-1) + u(k) \qquad (5-14)$$

it will be a wide band system. The simulation results for the wide band system are shown from Fig-5.33 to Fig-5.37. The phase portrait is shown in Fig-5.33 and is much more wide spread than Fig-5.28. After 99,950 learning iterations, the curve of d(x) is shown in Fig-5.34. When compared with Fig-5.30, it is quite obvious that the small discrepency region is much bigger. The tracing performance during the learning period is shown in Fig-5.35. Fig-5.36 shows the tracing under a square wave excitation after learning. The square wave is

$$u(k) = \begin{cases} -0.4, & \text{if } 100 \times n \leq k \leq 100 \times n + 50; \\ 0.4, & \text{otherwise.} \end{cases}$$

The two output traces are almost identical in Fig-5.36. Clearly the generalization is better.

Now we consider the nonlinear system situation. It is assumed that the nonlinear function f( ) in the equation (5-7) is bounded. In this case, it can be proved

that any state of the nonlinear system which is described by the equation (5-7) can be reached from any other state in finite time with an appropriate input excitation. For a nonlinear system with order of N, the equation (5-7) can be rewritten in a state variable form as

$$\begin{bmatrix} x(k+1) \\ x(k) \\ \vdots \\ x(k-N+2) \end{bmatrix} = \begin{bmatrix} f(x(k),\ldots,x(x-N+1)) \\ x(k) \\ \vdots \\ x(k-N+2) \end{bmatrix} + \begin{bmatrix} u(k) \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad (5-15)$$

If at time k, the state of the nonlinear system is $\vec{X}^o$, that is

$$\vec{X}_k = \begin{bmatrix} x(k) \\ \vdots \\ x(k-N+1) \end{bmatrix} = \begin{bmatrix} x_1^o \\ \vdots \\ x_N^o \end{bmatrix} = \vec{X}^o$$

and the destination state is $\vec{X}^d = [x_1^d, \ldots, x_N^d]^T$, then the excitation u(k) can be designed in the following way to make $\vec{X}_{k+N} = \vec{X}^d$. Now let

$$\begin{aligned} u(k) &= x_N^d - f(x_N^o, \ldots, x_1^o) \\ &= x_N^d - f(x(k), \ldots, x(k-N+1)) \end{aligned}$$

then we have $x(k+1) = x_N^d$. If we let

$$u(k+1) = x_{N-1}^d - f(x(k+1), \ldots, x(k-N))$$

then $x(k+2) = x_{N-1}^d$, and go on untill

$$u(k+N-1) = x_1^d - f(x(k+N-1), \ldots, x(k))$$

then

$$\vec{X}_{k+N} = \begin{bmatrix} x(k+N) \\ \vdots \\ x(k+1) \end{bmatrix} = \begin{bmatrix} x_1^d \\ \vdots \\ x_N^d \end{bmatrix} = \vec{X}^d$$

Now we have shown that any state can be reached from any other state in finite time with an appropriate excitation. That means the nonlinear system described by the equation (5-7) is controllable and every corner of the state space is reachable.

The controllability of the system can only guarantee theoretically that the phase portrait can cover the whole state space, but as shown in the linear system case, the phase portrait of the system under a general excitation is determined by the dynamic property of the system. For the linear system, it is the bandwidth which influences the phase portrait. The bandwidth of the system determines the transient response of the system. A wide band system has a quick decay transient response, so under an external excitation the output is less influnenced by the past experence. Thus the correlation is weak and the phase portrait is usually wide spread. In contrast for a narrow band system, because of its long lasting transient response, the correlation is strong and the phase portrait is more likely to be concentrated in the diagonal region. Although the concept of bandwidth and transient response cannot be applied to the nonlinear system directly, it is still reasonable to conclude from the foregoing analysis of the linear system that the phase portrait of the nonlinear system is influenced by the strength of its equilibrium attractor. For a strong attractor, the output of the system will have weak correlation and the phase portrait is wide spread. For a weak attractor, the phase portrait will be more likely to concentrate in the diagonal region. The strong attractor here means that any deviation from the equilibrium point will be attracted back very quickly. It is similar to the fast decay transient response in the linear system.

To verify the above prediction, three different nonlinear systems have been simulated. System A can be represented by the equation

$$x(k+1) = 1.8sat(x(k)) - 0.9sat(x(k-1)) + u(k) \qquad (5-16)$$

where sat( ) is a linear saturation function which has the form

$$sat(x) = \begin{cases} 0.9, & \text{if } x > 0.9; \\ x, & \text{if } -0.9 < x < 0.9; \\ -0.9, & \text{if } x < -0.9. \end{cases}$$

system B can be described by the equation

$$x(k+1) = \frac{x(k)x(k-1)(x(k)+2.5)(x(k)-1)}{1+x^2(k)+x^2(k-1)} + u(k) \qquad (5-17)$$

and system C is described by

$$x(k+1) = \frac{0.827x(k)(1-x(k-1))}{1+x^2(k)+x^2(k-1)} + u(k) \qquad (5-18)$$

All these three systems have a uniformly asymptotically stable equilibrium at $x = 0$. Fig-5.37a, Fig-5.37b and Fig-5.37c show how following a deviation the system returns to the equilibrium state for three different systems. The phase portraits of these three systems are shown in Fig-5.38a, Fig-5.38b and Fig-5.38c respectively. System A has the longest transient process (see Fig-5.37a) its output is strongly correlated and the phase portrait is concentrated in a narrow diagonal region (see Fig-5.38a). System B has the shortest transient process (see Fig-5.37b), so its phase portrait exhibits the greatest spread (see Fig-5.38b). The identification simulation results are shown in Fig-5.39 to Fig-5.44. Fig-5.39 shows the output traces of the system A and its neural network model. It may be seen there is rarely any dramatic change and thus the correlation is strong. Fig-5.40 shows the output traces of system B and its neural network model, the correlation is much weaker. From Fig-5.39 to Fig-5.41, it is clear that the tracing performance of the neural system is good in all these cases. But the identification performance of the system A and system C is not satisfactory as shown by the d(x) curves in Fig-5.42 and Fig-5.44. Because of their narrowly spread phase portraits, the small error regions are also small. The d(x) curve of system B is shown in Fig-5.43, it has a larger small error region. In summary, to obtain a satisfactory identification of a system with a MLP neural network, the system needs to have a short transient process. This is true for both linear and nonlinear systems.

5.3.2 Prediction of Chaotic Time Series

The above discussion about the application of MLP neural networks to nonlinear system identification is restricted to systems which have an asymptotically stable equilibrium point. As is well known there are a large number of nonlinear systems whose attractors are not simply points or limit cycles, but are strange attractors which can lead the system into chaotic behaviour[112],[113]. It is a natural extension to discuss the identification of nonlinear systems which have strange attractor structure with MLP neural networks. Generally speaking, a strange attractor is an assembly of an infinite number of points which are the states of a autonomous dynamic chaotic system. A mathematical explanation of strange attractors can be found in [113]. For the nonlinear system described by the equation

$$x(k+1) = (x(k) - 1.3)(x(k) + 1.1)(x(k-1) - 1.1)(x(k-1) + 0.9) + u(k) \quad (5-19)$$

if $u(k) = 0$, it has a strange attractor shown in Fig-5.45a. This kind of phase portrait is obviously unfavourable for identification with an MLP neural network. If u(k) is a random excitation, the phase portrait will be more wide spread, which is shown in Fig-5.45b. The identification simulation results are shown in Fig-5.46 to Fig-5.48. Fig-5.46 shows the d(x) curve obtained after 399,950 learning iterations under zero excitation. In this case although the output of the nonlinear system looks random, its phase portrait is restricted to the strange attractor shown in Fig-5.45a and the identification has failed. Fig-5.47 shows the d(x) curve obtained after 399,950 learning iterations under random excitation and clearly the identification is much better. However the real problem with the identification of chaotic systems is that, for a chaotic system any infinitesimally different starting points will produce significantly different outcomes. So any small modeling error will be amplified to its maximum in the dynamic process. The d(x) curve in Fig-5.47 shows that the neural network approximates the chaotic system quite reasonably in the central

region of the state plane. However when the neural system and the chaotic system are started from the same initial point close to the origin of the state plane, the dynamic processes shown in Fig-5.48 are totally different after a few steps. In this sense, the identification has failed. This represents a fundamentally difficult problem for the identification of a chaotic system with an MLP neural network.

One of the objectives of chaotic system research is to predict the development of some random like processes. (e.g turbulence, population dynamic process in Ecology, and climate dynamic processes.) The pratical problem of prediction is to use the past sample data to predict the future development of the process. For a autonomous chaotic system represented by the equation

$$x(k+1) = f(x(k), x(k-1), \ldots, x(k-n+1))$$

its MLP neural network predictor can be formed like

$$\hat{x}(k+p) = NN(x(k), x(k-1), \ldots, x(k-n+1))$$

where NN( ) is a MLP neural network and p is the forward prediction step. It is impractical to do long-term prediction about chaotic time series because of the reason discussed in the above paragraph. However MLP network can be used quite successfully for short-term prediction. A one step forward prediction simulation is shown in Fig-5.49. From Fig-5.48 we can see that after about 10 steps the match between the chaotic system and MLP network model breaks down. So for this system, the feasible predication range is around ten steps.

As the phase portrait of a chaotic system is restricted to a strange attractor in its phase space, if we have a reasonable long observation, we can always obtain a learning sample set which gives a typical representation of the strange attractor. Then a MLP network can be trained to perform prediction. This feature of phase

portrait of chaotic time series was explored in some other prediction algorithms [114].

5.4 Concluding Remarks

In conclusion, the following points should be noted.

• It may appear that the nonlinear system identification with MLP networks is the same as parameter or coefficient estimation if we regard the weights of the network as parameters. However there are some difference. In parameter estimation we usually hope the parameters converge to a unique solution, while in MLP network identification, we are not concerned with the value of the weights. Due to the multi-solution feature of MLP networks the weights can take any value as long as the overall input-output relationship is a good approximation of the mapping we are modeling. In addition, the MLP network has a more powerful representation ability than usual parameter models.

• For the identification of systems with an MLP neural network, random excitation is usually needed. This is not only for the coverage of the learning samples, but also for the convergence of the weights in the model-I case.

• As has already been shown the universality of a neural network does not necessarily give it advantages in applications. In the identification problem, this principle has been illustrated again. For the linear systems discussed in section 3, a single layer linear perceptron can identify the system under a less restricted condition and use shorter learning time than the MLP neural network, and can also give a better generalization, although the MLP neural network has a more powerful representation ability. The match between the built-in structure of the neural

network and that of the system to be modeled is vital for satisfactory identification and efficient learning. But in the case when little structure information is available, the MLP neural network model can always be used as a last resort.

- To use the MLP neural network for system identification, the system to be identified and excitation should meet certain conditions, otherwise the identification performance will be very poor.

- If the identification is only restricted to a small part of the state space, theoretically we cannot say the identification is completed. However from a practical point of view, the results may still have application value. For example in the narrow band system, correct identification is restricted in the diagonal region, but under general conditions the phase portrait of the system will rarely go out of this range. To drive the phase portrait out of the diagonal region, a strong high frequency excitation is required and would rarely occur in a practical situation. In some cases, although the neural system is a poor model of the real system, it may still be a good predictor. The chaotic time series prediction discussed in section 3 is an example.

In the next chapter we discuss the application of artificial neural networks in communication channel equalization.

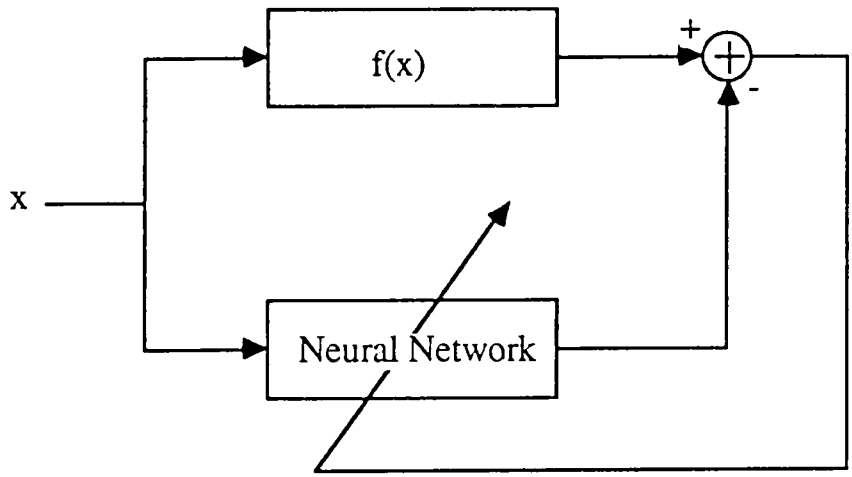


Fig-5.1 Static Mapping Identification

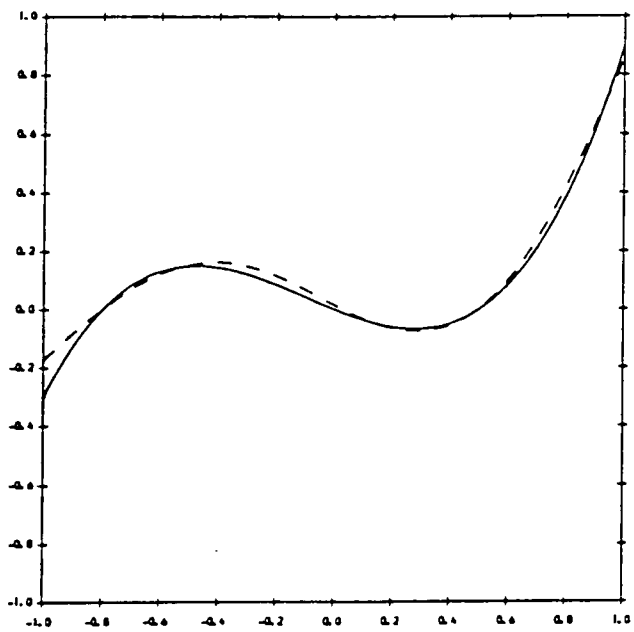


Fig-5.2

Solid line is the curve of f(x) and the broken line is of the neural network. learning time is 50000, learning section is [-1, 1], step size=0.25

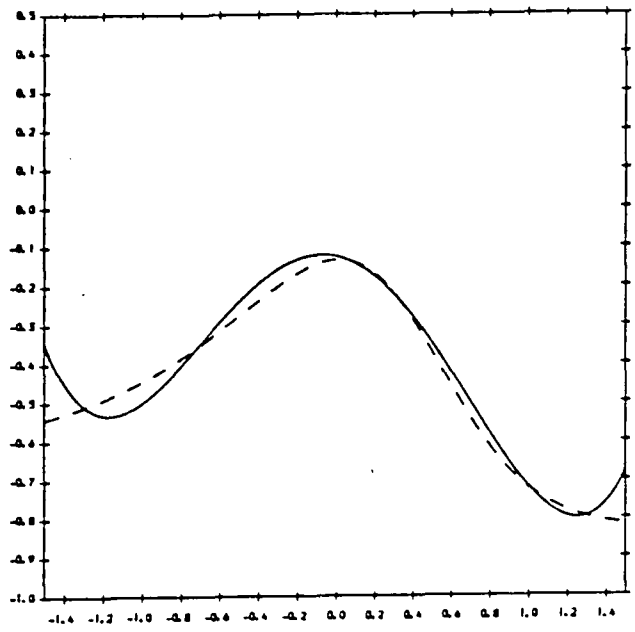


Fig-5.3

Solid line is the curve of f(x) and the broken line is of the neural network. learning time is 50000, learning section is [-1.5, 1.5], step size=0.25.

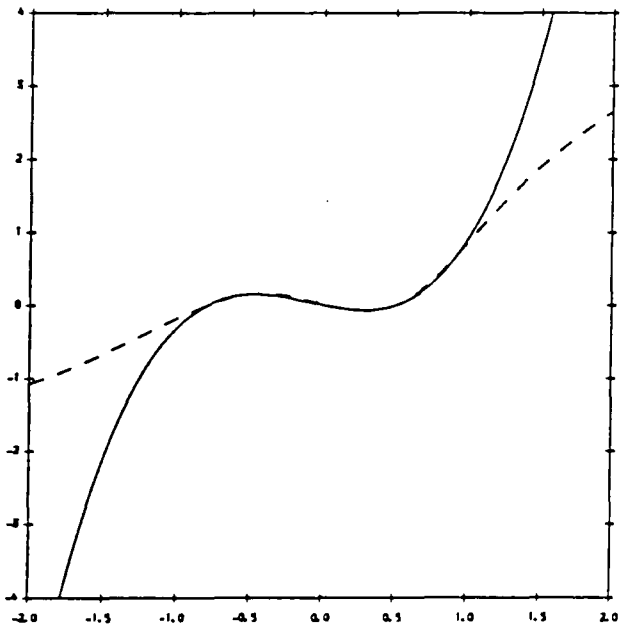


Fig-5.4

Solid line is the curve of f(x) and the broken line is of the neural network. learning time is 50000, learning section is [-1, 1], step size=0.25.

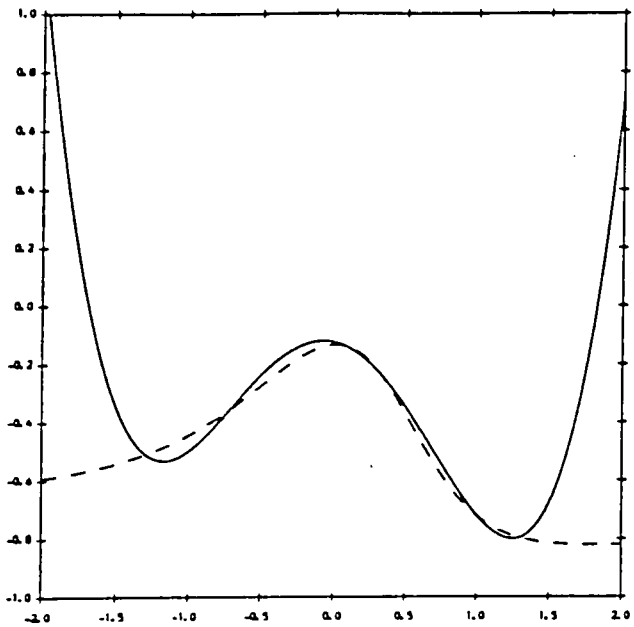


Fig-5.5

Solid line is the curve of f(x) and the broken line is of the neural network. learning time is 50000, learning section is [-1.5, 1.5], step size=0.25.

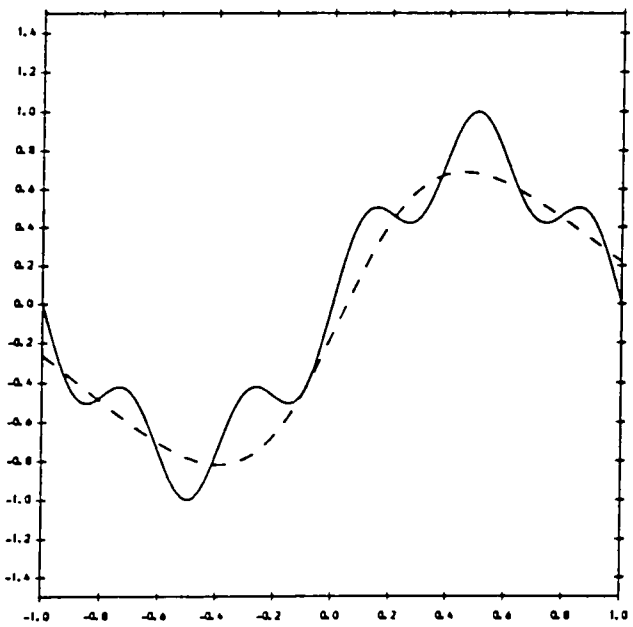


Fig-5.6

Solid line is the curve of f(x) and the broken line is of the neural network. learning time is 50000, learning section is [-1, 1], step size=0.25.

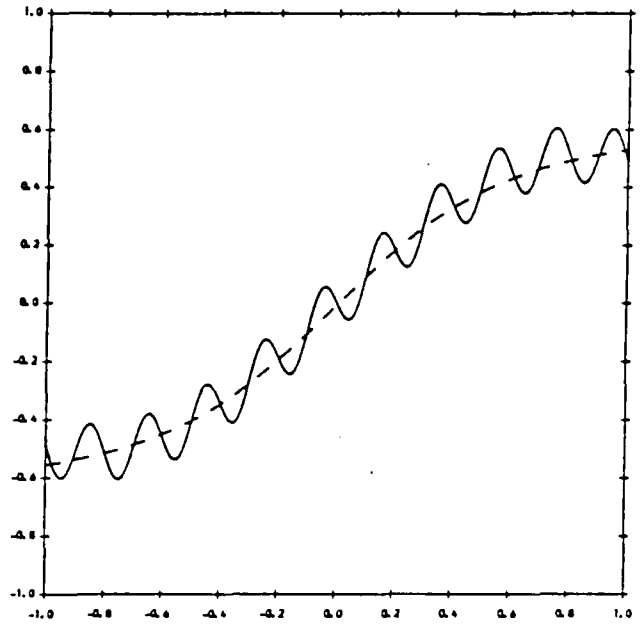


Fig-5.7

Solid line is the curve of f(x) and the broken line is of the neural network. learning time is 50000, learning section is [-1, 1], step size=0.25.

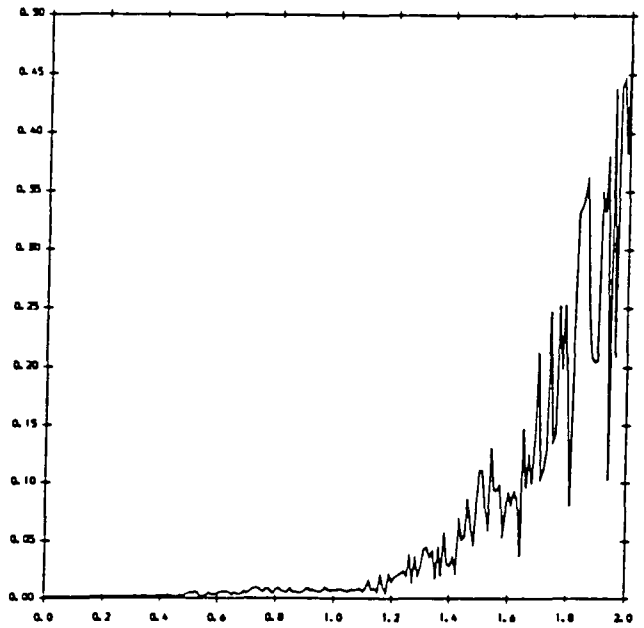


Fig-5.8

The curve of d(x) on [0, 2]. The learning time in this case is 300,000. The learning area is defined by $[-1 < x_1 < 1, \ and \ -1 < x_2 < 1]$, and the step size is 0.25.

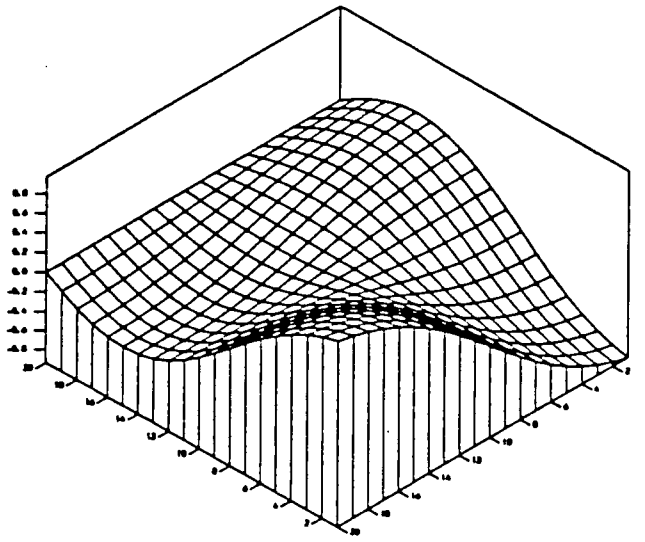


Fig-5.9

The surface of the $f(x_1, x_2)$. The displaying region is $[-1 < x_1 < 1, \ and \ -1 < x_2 < 1]$. Detail see Fig-5.8 and the text.

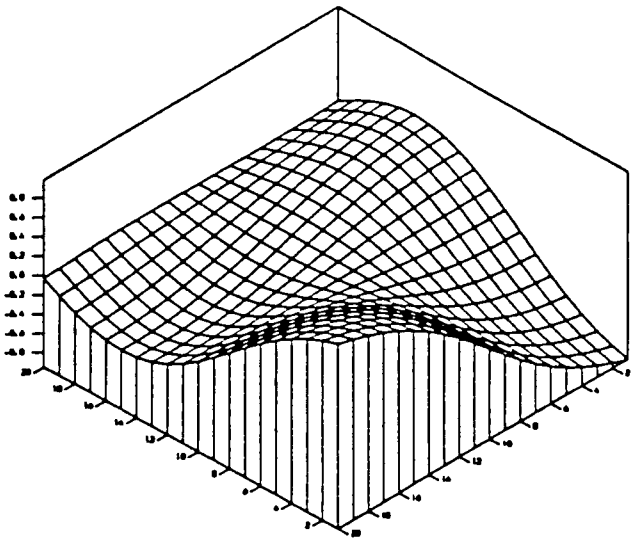


Fig-5.10

The surface formed by the neural network. The displaying region is $[-1 < x_1 < 1, \ and \ -1 < x_2 < 1]$. Detail see Fig-5.8 and the text.

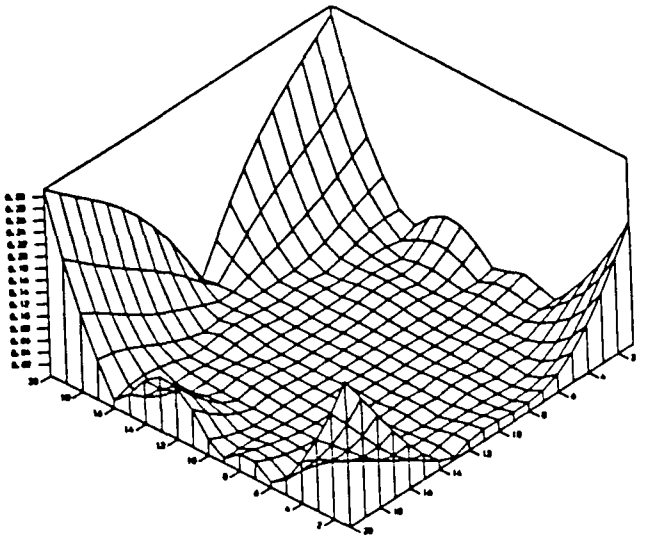


Fig-5.11

The error surface between the function $f(x_1, x_2)$ and the neural network. The displaying region is $[-1.5 < x_1 < 1.5, \ and \ -1.5 < x_2 < 1.5]$. Detail see text.

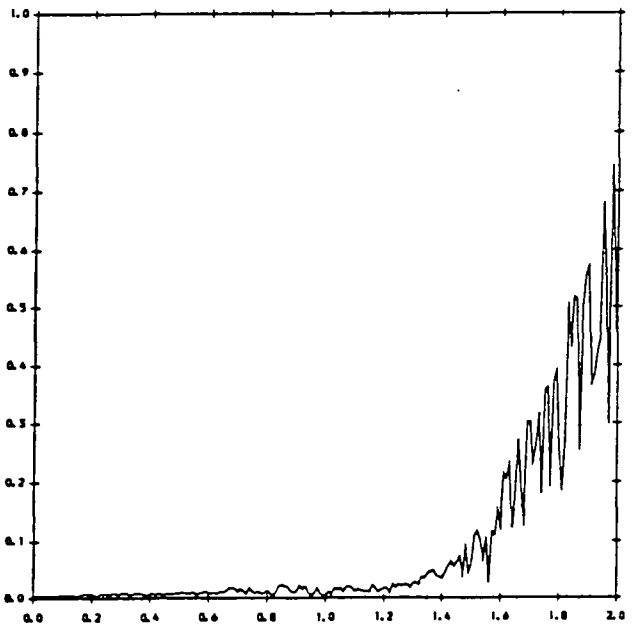


Fig-5.12

The curve of d(x) on [0, 2]. The learning time in this case is 200,000. The learning area is defined by $[-1 < x_1 < 1, \ and \ -1 < x_2 < 1]$, and the step size is 0.25.

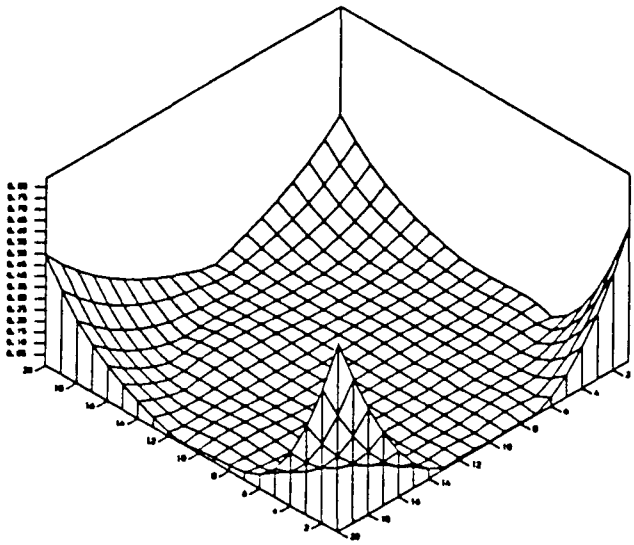


Fig-5.13

The error surface for surface approximating problem described in Fig-5.12 and the display region is $[-1.5 < x < 1.5, \quad -1.5 < y < 1.5]$.

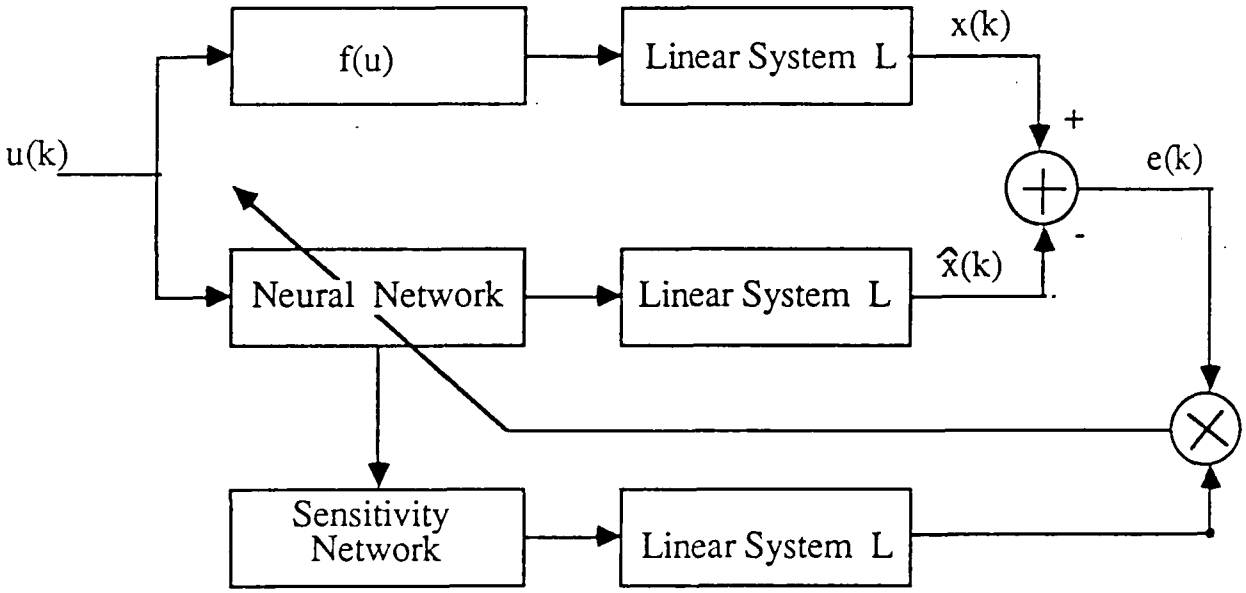


Fig-5.14 Model-I Nonlinear System Identification Scheme

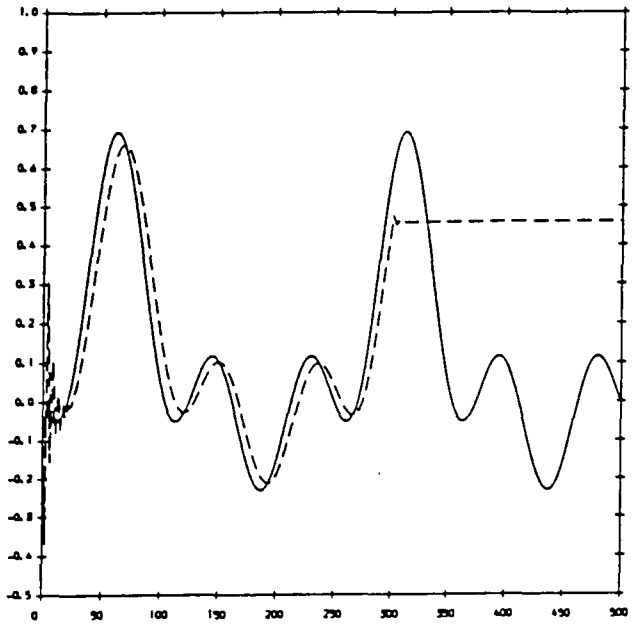


Fig-5.15

The solid line is the output of the nonlinear system to be identified, and the broken line is the output of the neural system. Weight updating stopped at time 300. $a_1 = 0.3$, $a_2 = -0.6$, step size=0.25, the excitation is $u(k) = sin(\frac{2\pi k}{250})$

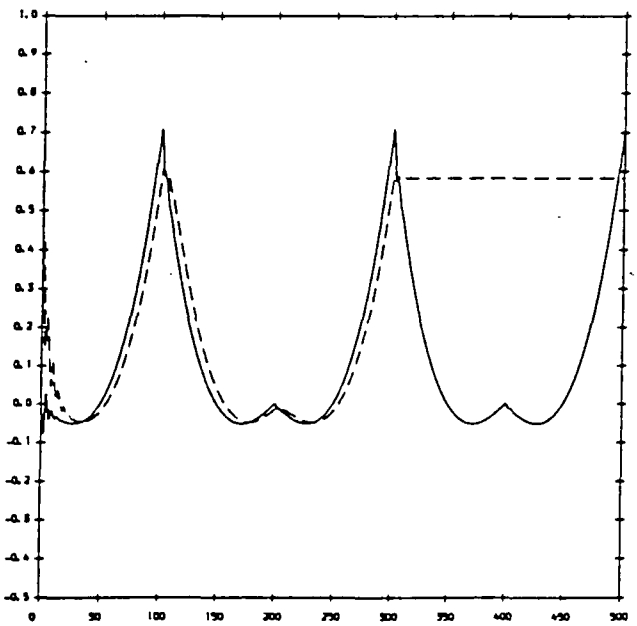


Fig-5.16

The solid line is the output of the nonlinear system to be identified, the broken line is the output of the neural system. The simulation condition is the same as in Fig-5.15, except that the excitation is the triangular wave (or saw tooth wave).

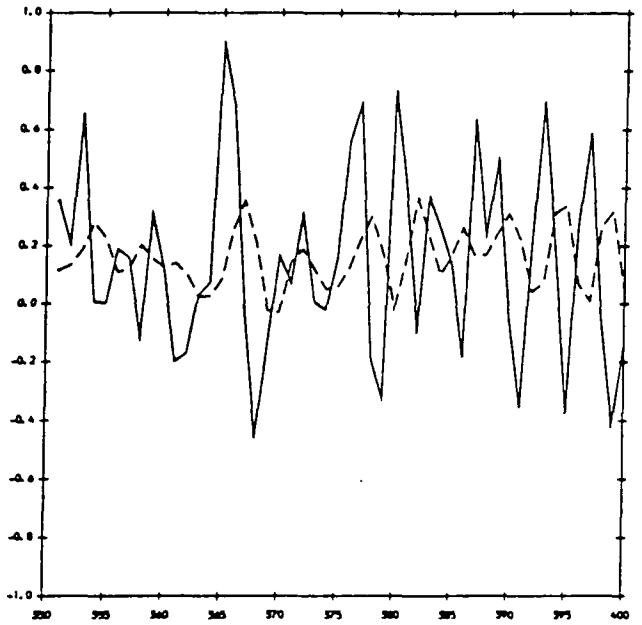


Fig-5.17

The solid line is the output of the nonlinear system to be identified, the broken line is the output of the neural system. The simulation conditions are the same as in Fig-5.15, except that the excitation is a random process with an uniform distribution on [0, 1].

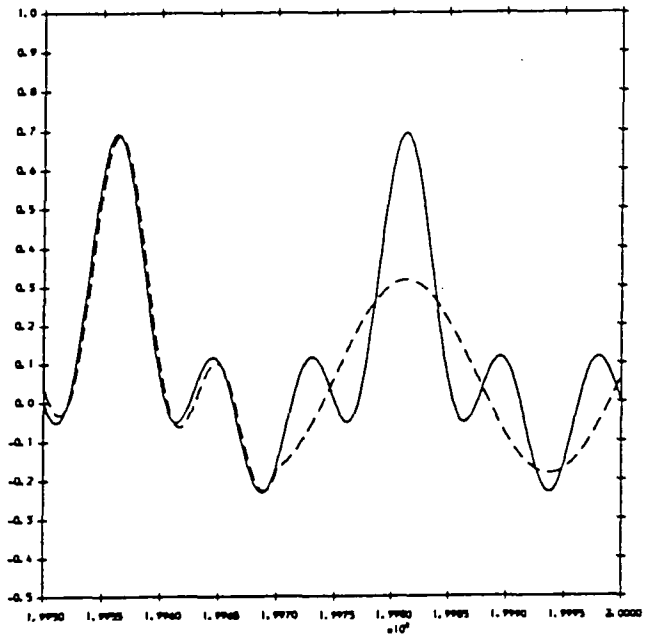


Fig-5.18a

The solid line is the output of the nonlinear system to be identified, the broken line is the output of the neural system. The simulation conditions are the same as in Fig-5.15, except that the learning stopped at time 199700.

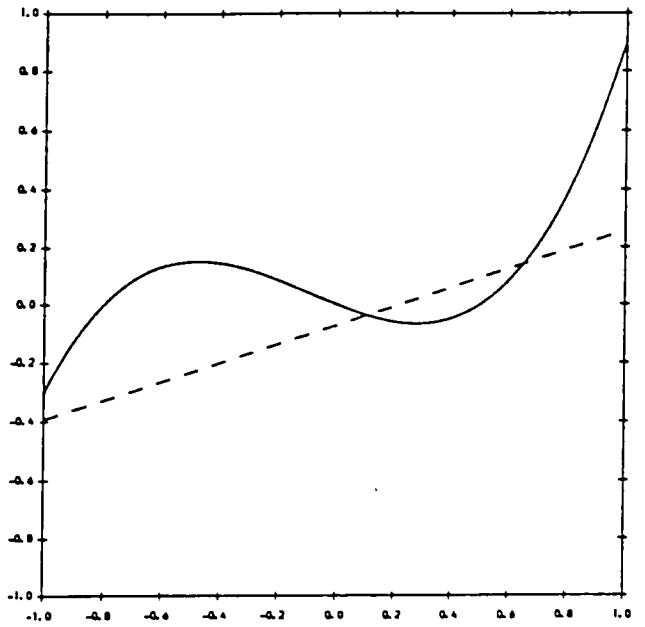


Fig-5.18b

The solid line is the curve of the nonlinear function f(u) for $u \in [-1, 1]$, and the broken line is the curve of the 1-20-10-1 neural network in [-1, 1]. The simulation conditions are the same as in Fig-5.18a

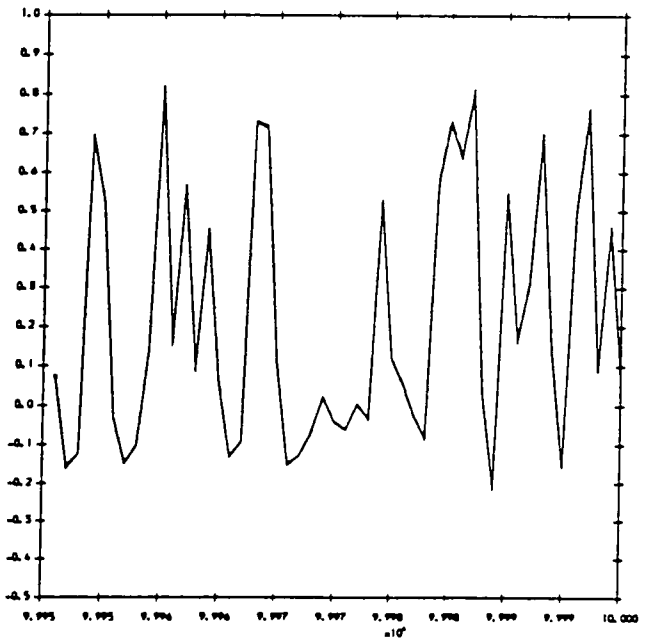


Fig-5.19a

The solid line is the output of the nonlinear system to be identified, the broken line is the output of the neural system. Actually, they are fitted together. In this case, the learning stopped at time 99800, $a_1 = 0.3$, $a_2 = -0.2252$, step size=0.25, the excitation is a random process with an uniform distribution on [0, 1].

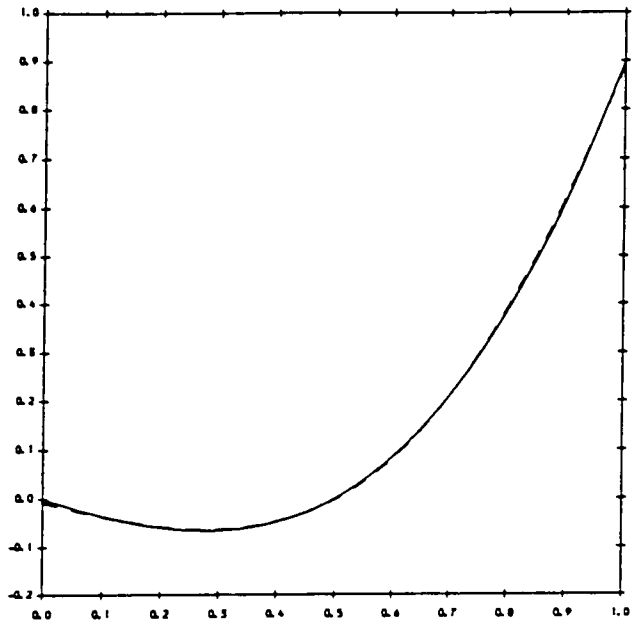


Fig-5.19b

The solid line is the curve of the nonlinear function f(u) for $u \in [-1, 1]$, and the broken line is the curve of the 1-20-10-1 neural network in [-1, 1]. The simulation conditions are the same as in Fig-5.19a.

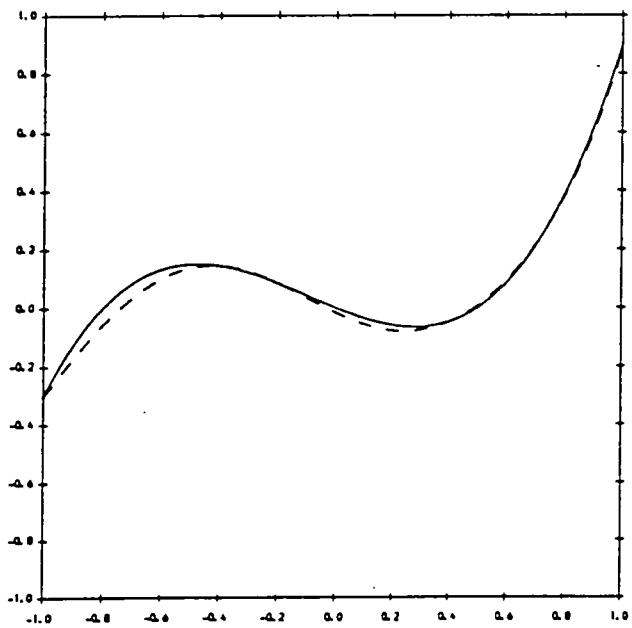


Fig-5.20

The solid line is the curve of the nonlinear function in [-1, 1], and the broken line is the curve of the 1-20-10-1 neural network in [-1, 1]. The simulation conditions are the same as in Fig-5.19a, except that the excitation is $u(k) = sin(\frac{2\pi k}{0.876513})$.

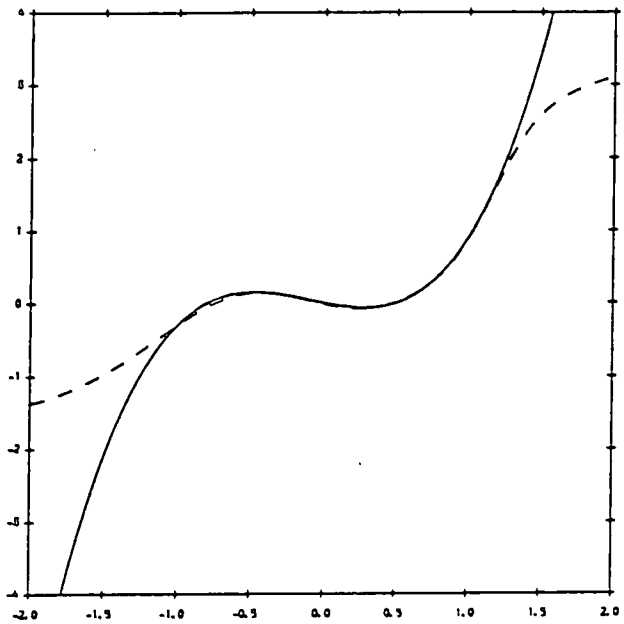


Fig-5.21

The solid line is the curve of the nonlinear function in [-2, 2], and the broken line is the curve of the 1-20-10-1 neural network in [-2, 2]. The simulation conditions are the same as in Fig-5.19a.

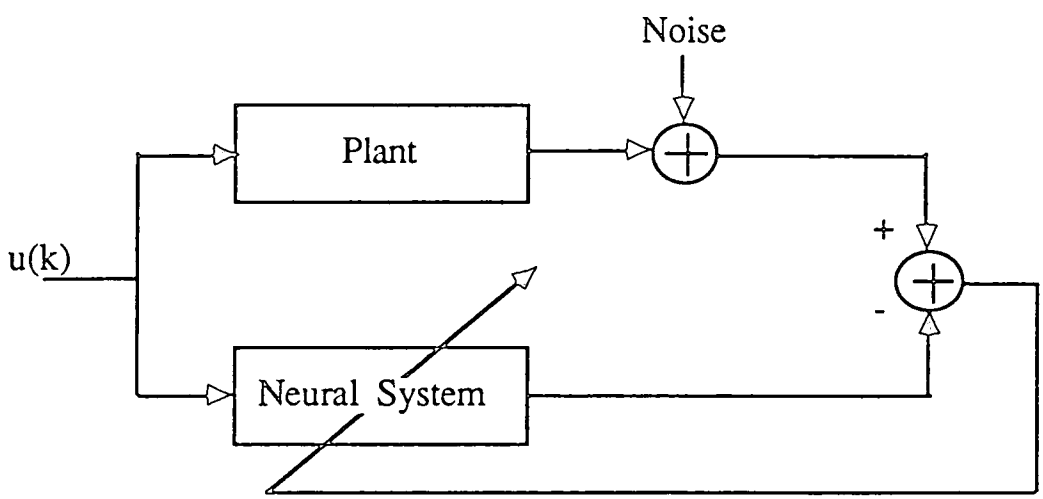


Fig-5.22 Output Noise Immunity Study

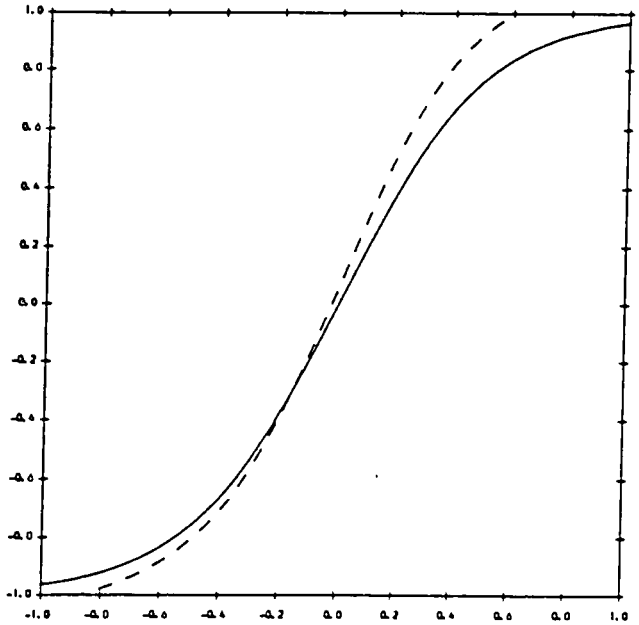


Fig-5.23

The solid line is the curve of the nonlinear function in [-1, 1], and the broken line is the curve of the 1-20-10-1 neural network in [-1, 1]. The simulation conditions are the same as in Fig-5.20, except that some noise with normal distribution is added. m=0.0, $\sigma = 0.333$, $\sigma^2 = 0.1$.

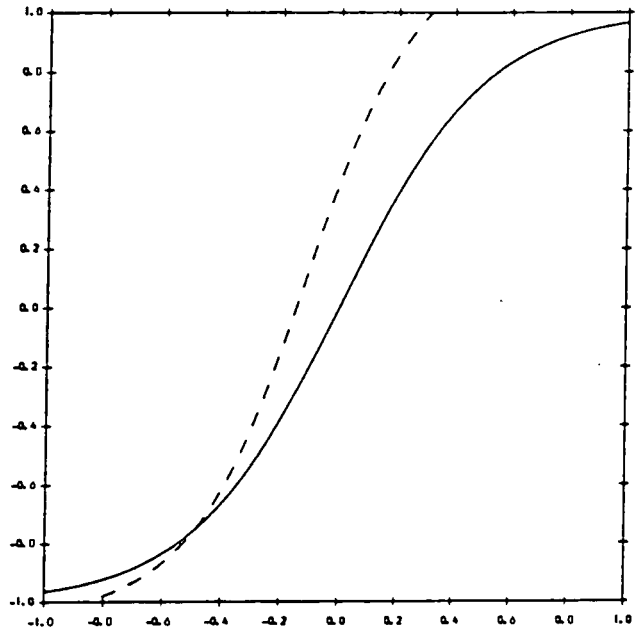


Fig-5.24

The solid line is the curve of the nonlinear function in -1, 1], and the broken line is the curve of the 1-20-10-1 neural network in [-1, 1]. The simulation conditions are the same as in Fig-5.23, except that $\sigma = 0.5$.

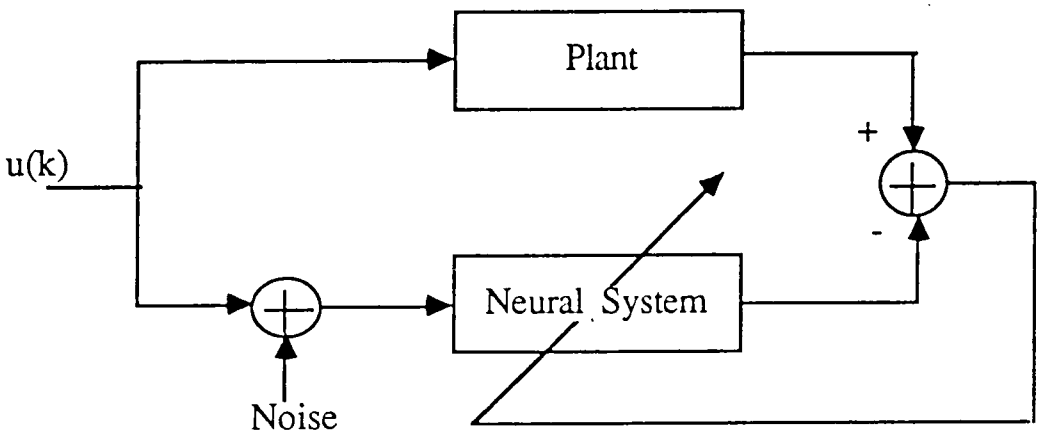


Fig-5.25 Input Noise Immunity Study

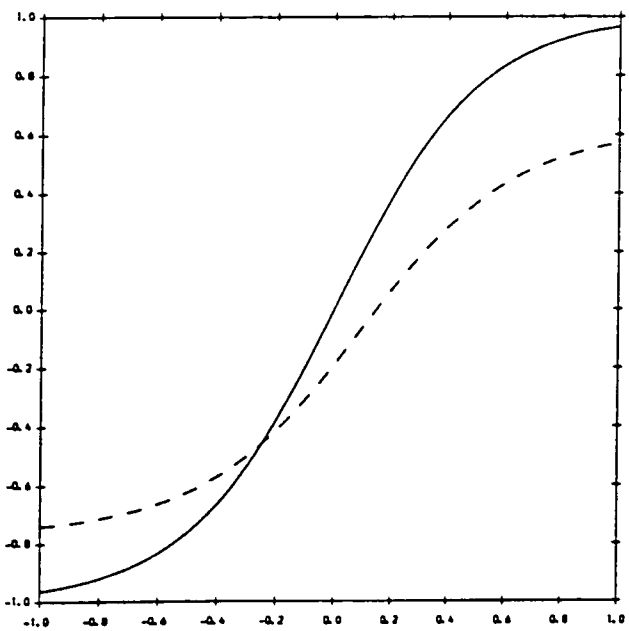


Fig-5.26

The solid line is the curve of the nonlinear function in [-1, 1], and the broken line is the curve of the 1-20-10-1 neural network in [-1, 1]. The simulation conditions are the same as in Fig-5.23, except that the noise is added at the input port of the neural system (see Fig-5.25).

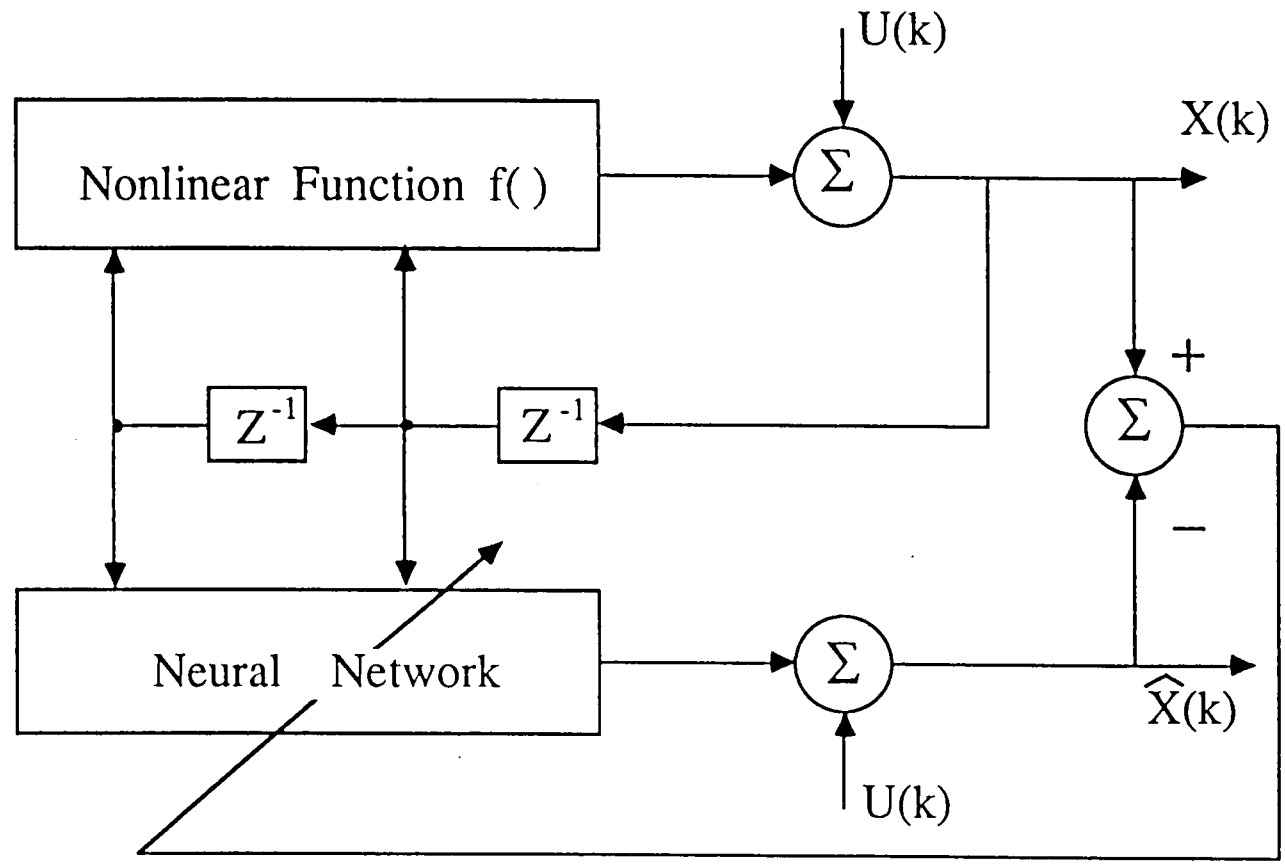


Fig-5.27 Serial Identification Architecture

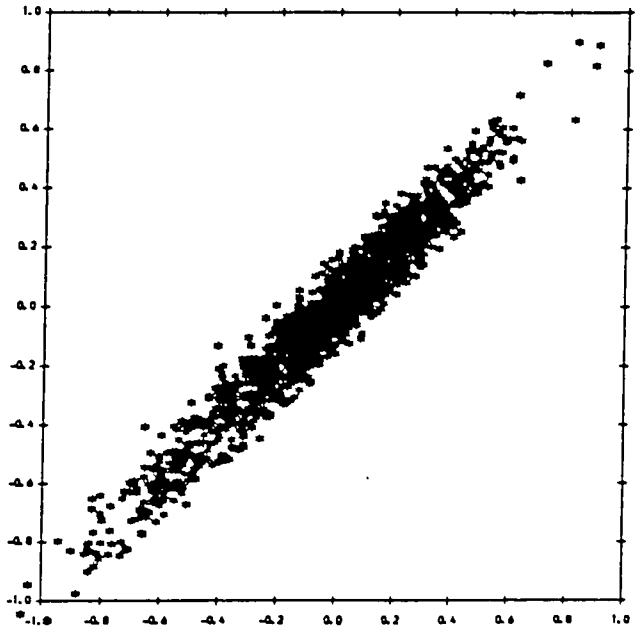


Fig-5.28

The phase portrait of the narrow band system which is described by the equation (5-13). The display region is $[-1 < x < 1, \quad -1 < y < 1]$.

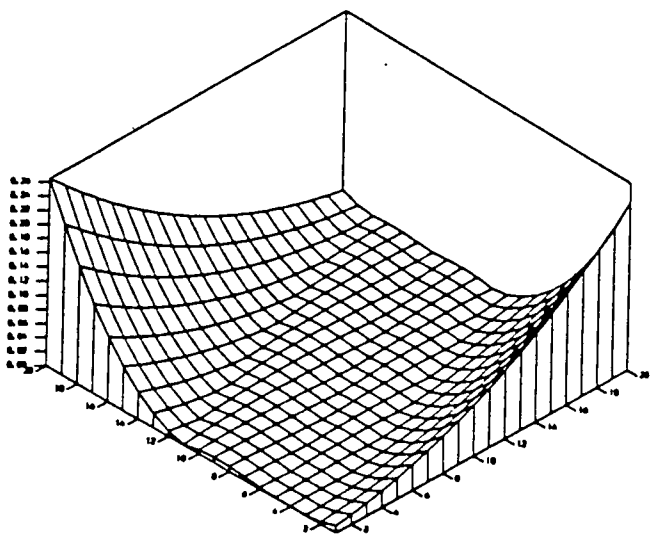


Fig-5.29

The error surface after 99,900 learning iterations. The excitation is a random process, the magnitude is uniformly distributed in [-0.175, 0.175]. The display region is $[-1 < x < 1, -1 < y < 1]$. The neural network is a 2-20-10-1 network. The step size=0.25

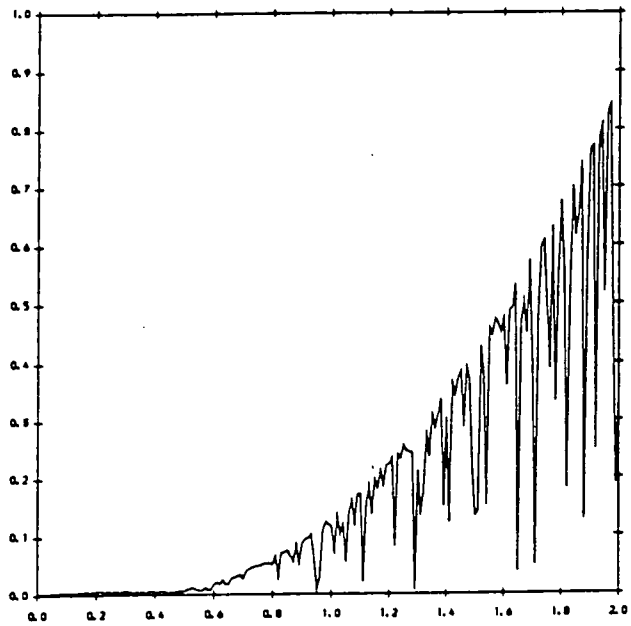


Fig-5.30

The curve of d(x). The learning conditions are the same as Fig-5.29.

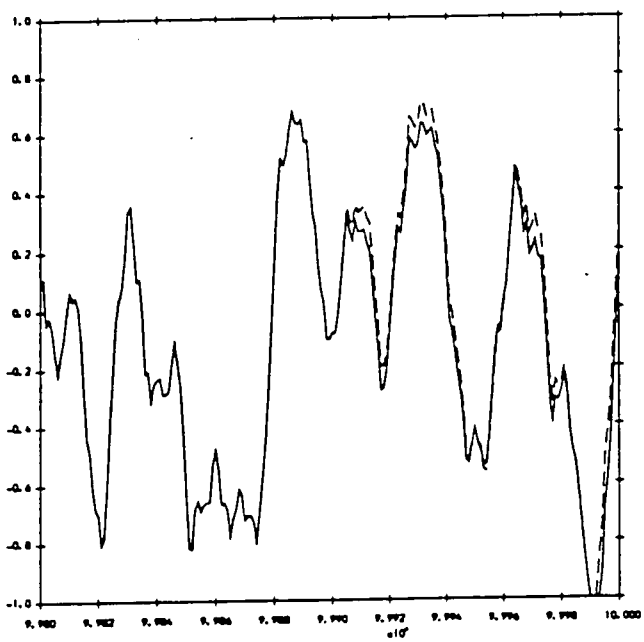


Fig-5.31

The broken line is the output trace of the neural system, and the solid line is the output trace of the plant. The learning stopped at iteration 99,900. The learning conditions are the same as in Fig-5.29.

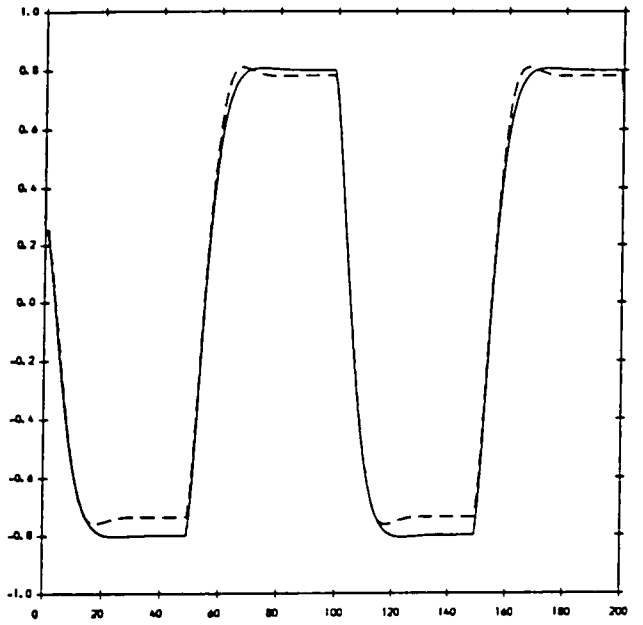


Fig-5.32

The broken line is the output trace of the neural system under a square wave excitation after 99,900 learning iterations, and the solid line is the output trace of the plant under the same excitation. The learning conditions see explanation of Fig-5.29.

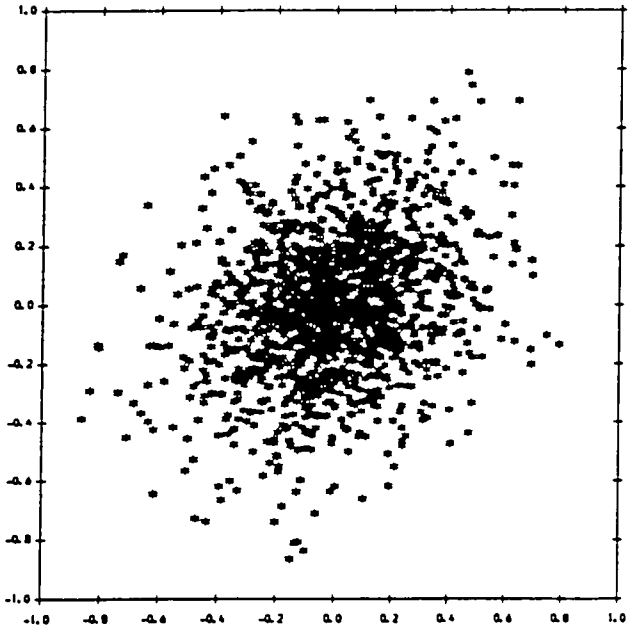


Fig-5.33

The phase portrait of the wide band system which is described by the equation (5-14).

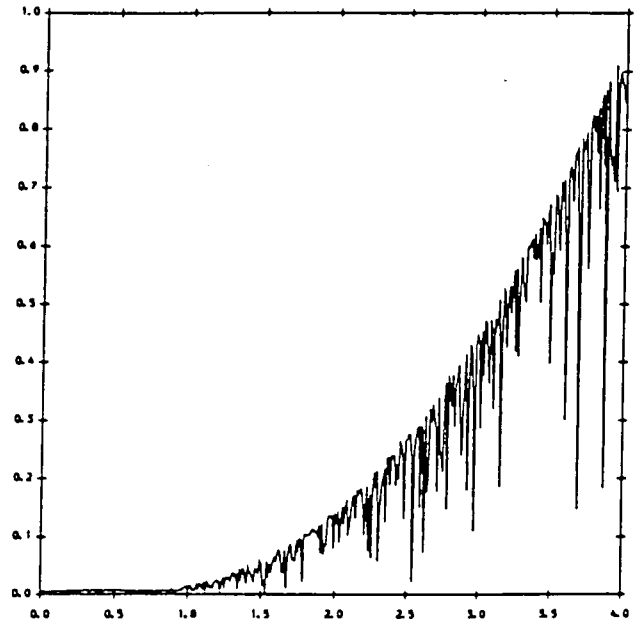


Fig-5.34

The curve of the d(x). The learning conditions see the explanation of Fig-5.35.

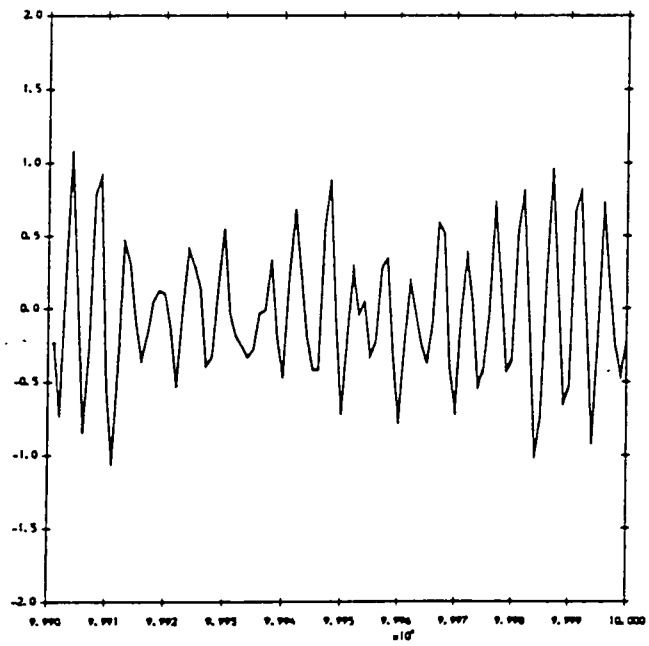


Fig-5.35

The broken line is the output trace of the neural system, and the solid line is for the plant. The learning stopped at iteration 99,950. A 2-20-10-1 network is used in this case, the excitation is a random process whose magnitude is uniformly distributed in [-0.4, 0.4], and the step size is 0.25.

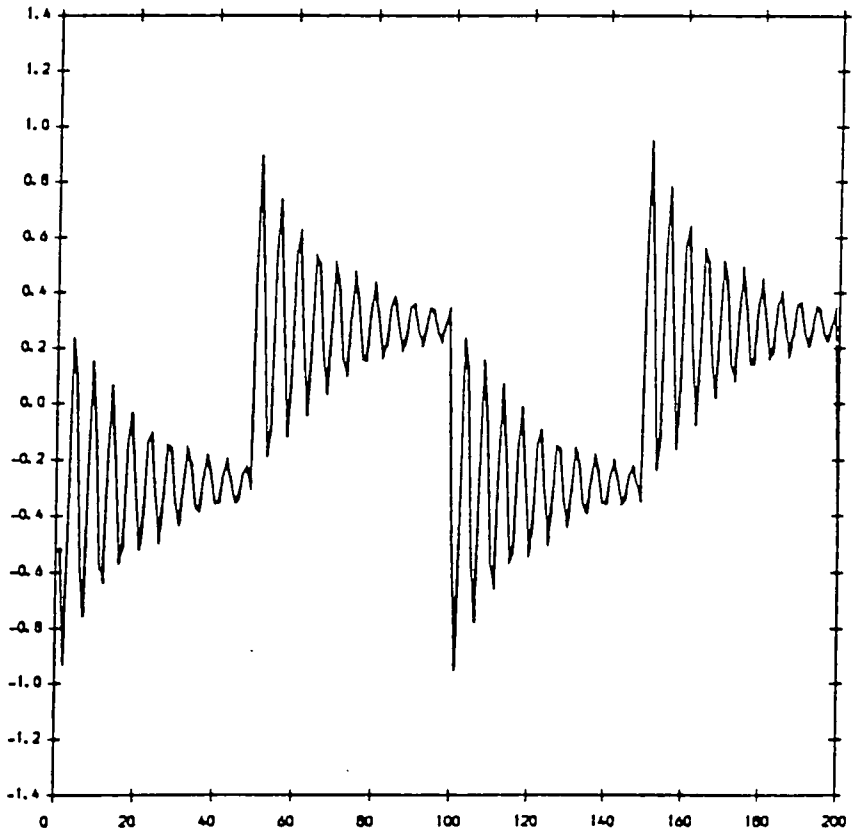
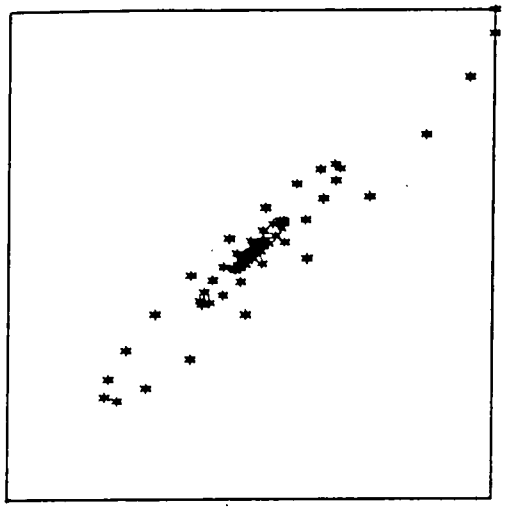
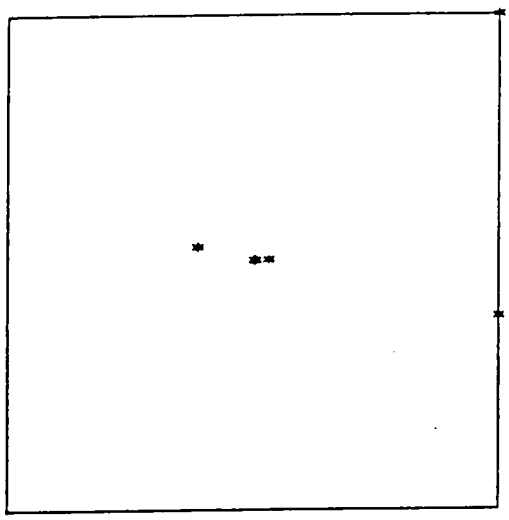


Fig-5.36

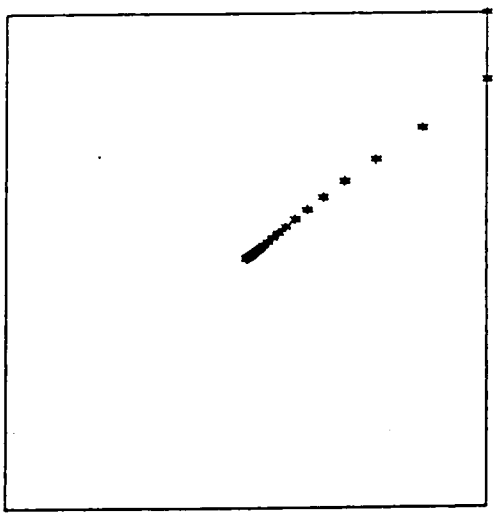
The broken line is the output trace of the neural system under a square wave excitation after learning, and the solid line is for the plant under the same excitation. For the learning conditions see the explanation of Fig-5.35.



(a)



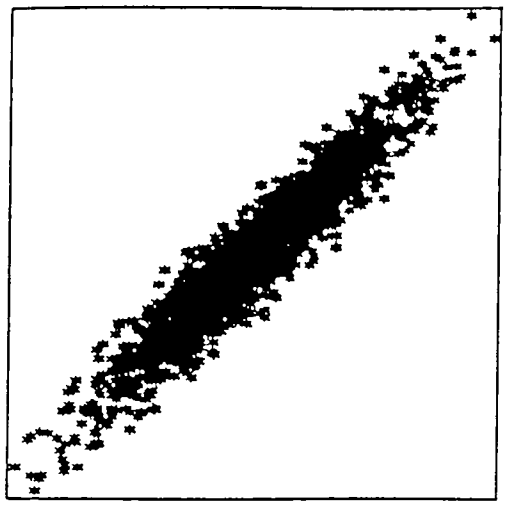
(b)



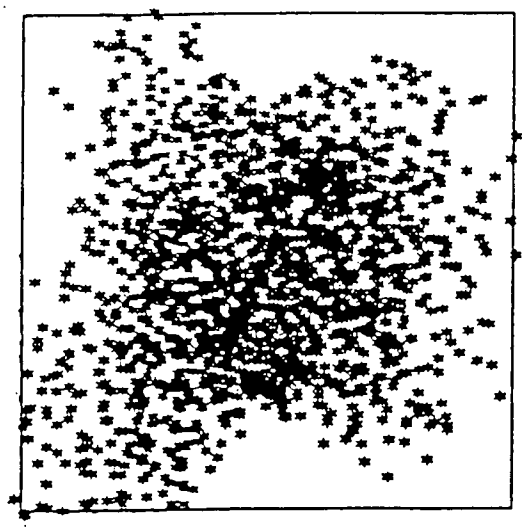
(c)

Fig-5.37

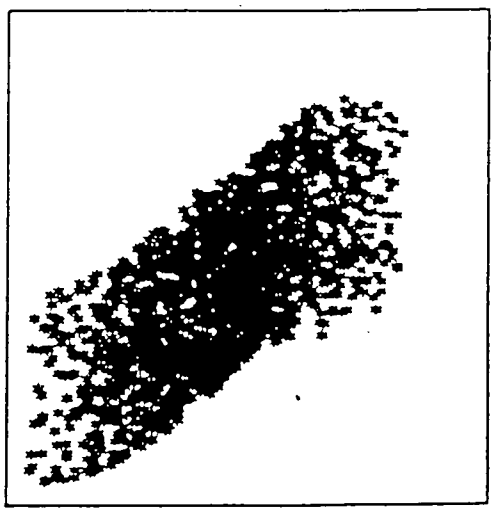
The transient process phase portraits of three nonlinear systems. The display region is $[-0.1 < x < 0.1, \ -0.1 < y < 0.1]$.



(a)



(b)



(c)

Fig-5.38

The phase portraits of the three nonlinear systems under random excitation. The display region is $[-1 < x < 1, \ -1 < y < 1]$.

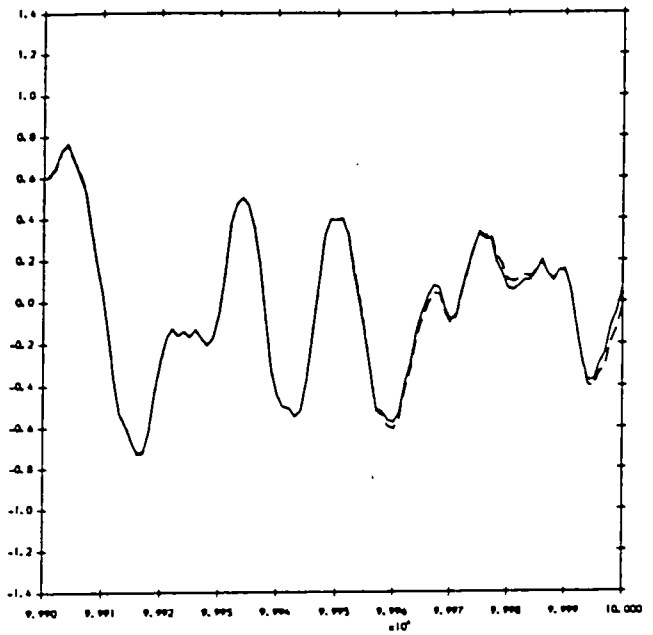


Fig-5.39

The output traces of system A and its neural model. The network used has 2-20-10-1 structure, the learning time is 99,950 and with the step size of 0.25. The random excitation distributed uniformly on [-0.075, 0.075]. The solid line is for the output of system A, and the broken line is for the neural system.

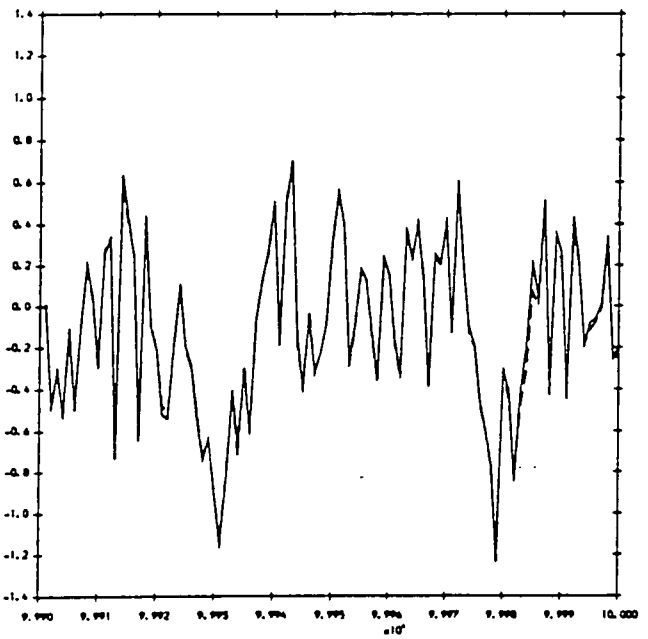


Fig-5.40

The output traces of system B and its neural model. The network used has 2-20-10-1 structure, the learning time is 99,950 and with the step size of 0.25. The random excitation distributed uniformly on [-0.6, 0.6]. The solid line is for the output of system B, and the broken line is for the neural system.

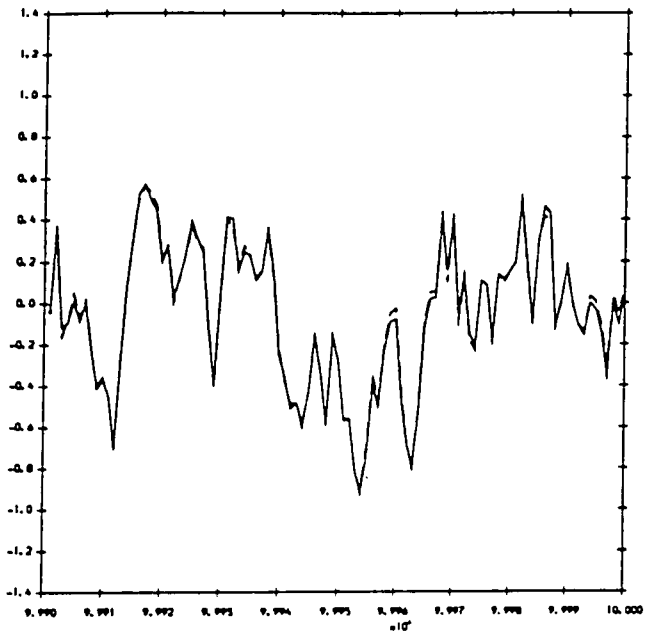


Fig-5.41

The output traces of system C and its neural model. The network used has 2-20-10-1 structure, the learning time is 99,950 and with the step size of 0.25. The random excitation distributed uniformly on [-0.4, 0.4]. The solid line is for the output of system C, and the broken line is for the neural system.

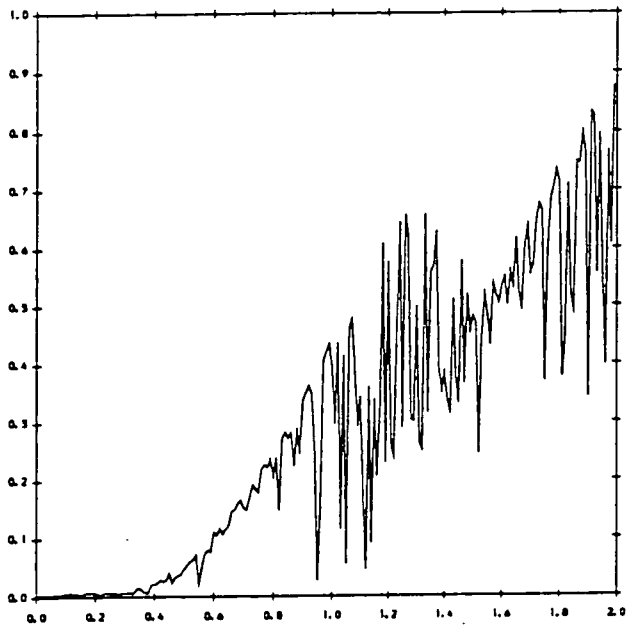


Fig-5.42

The d(x) curve of the neural system of system A. The learning conditions are the same as Fig-5.39.

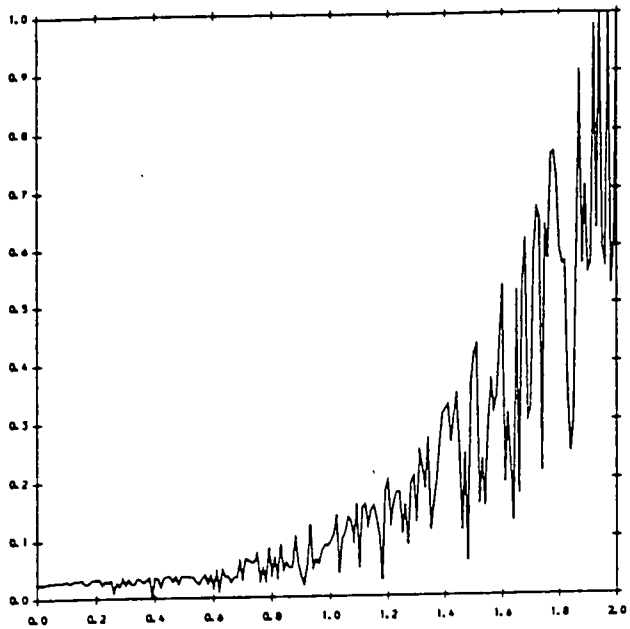


Fig-5.43

The d(x) curve of the neural system of system B. The learning conditions are the same as Fig-5.41.

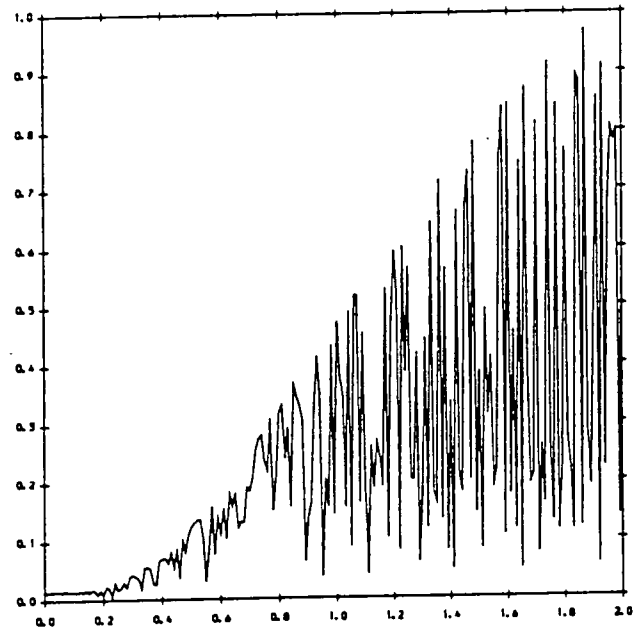
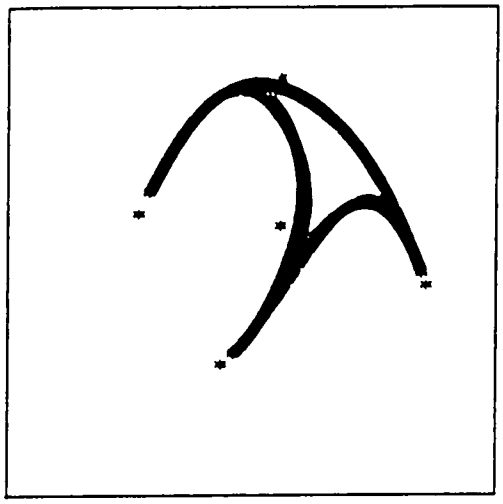
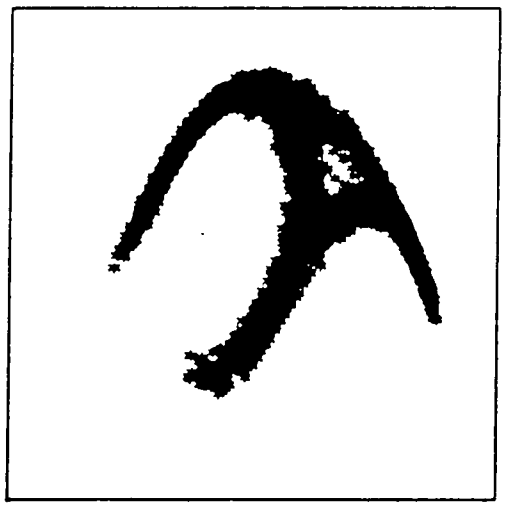


Fig-5.44

The d(x) curve of the neural system of system C. The learning conditions are the same as Fig-5.40.



(a)



(b)

Fig-5.45

The figure a is the strange attractor of a nonlinear system, and b is the phase portrait of the same system under a random excitation. The display region is $[-2 < x < 2, \quad -2 < y < 2]$.

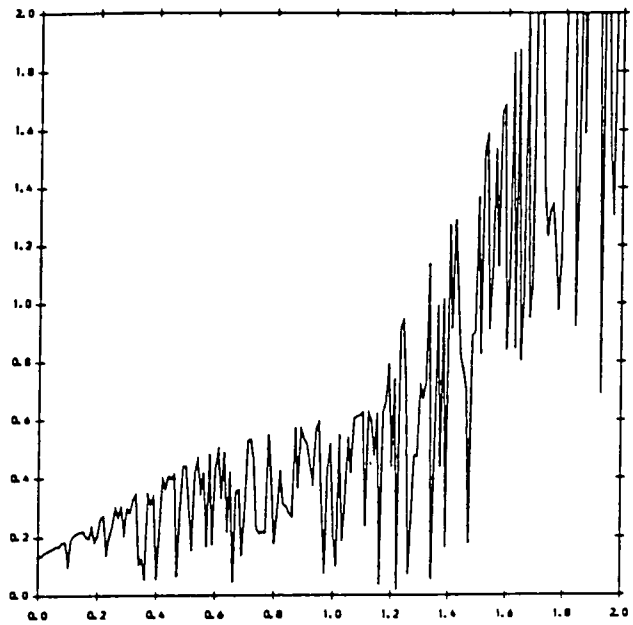


Fig-5.46

The d(x) curve of of the neural system. The neural network has the 2-20-10-1 structure, and the learning time is 399,950 with the step size of 0.25. The excitation is zero.

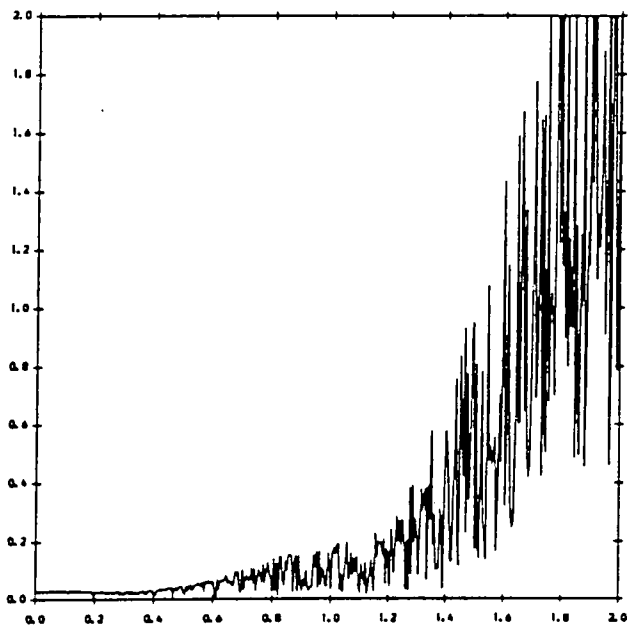


Fig-5.47

The d(x) curve of of the neural system. The neural network has the 2-20-10-1 structure, and the learning time is 399,950 with the step size of 0.25. The random excitation is distributed uniformly on [-0.1, 0.1].

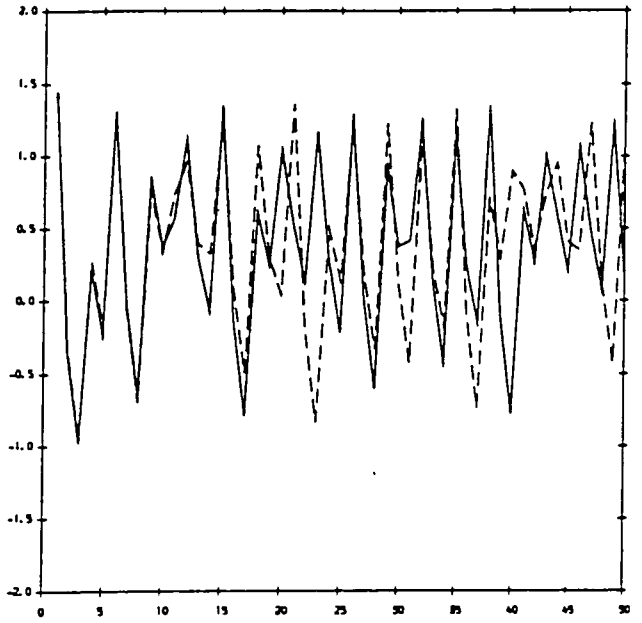


Fig-5.48

The output traces of the nonlinear system and its neural model after learning. The solid line is for the nonlinear system, and the broken line is for the neural system. The initial state is [0.2, 0.2]. The learning conditions are the same as Fig-5.47.

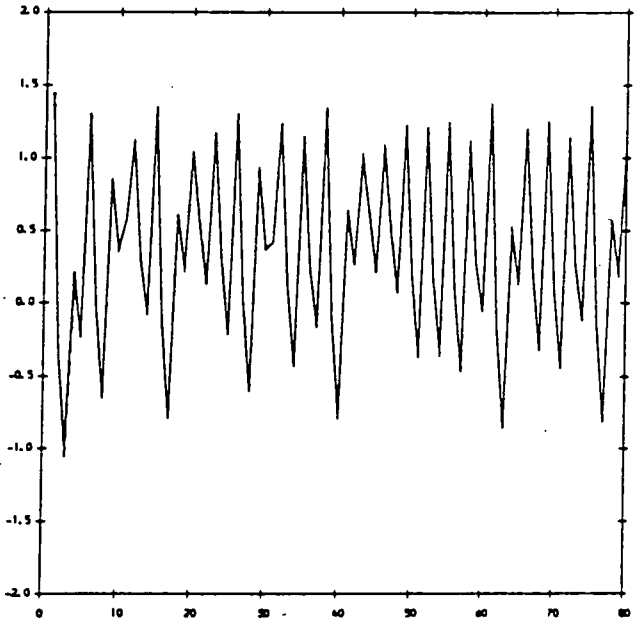


Fig-5.49

The output traces of the chaotic system and its predictor after learning. The solid line is for the chaotic system, and the broken line is for the predictor. The learning conditions are the same as Fig-5.46

Chapter Six

Adaptive Equalization Using Self-Organizing Neural Networks

6.1 Problem Definition

Intersymbol interference is one of the major practical problems in digital communication systems. This form of interference occurs when the signal to be transmitted has significant components of various frequencies at which the amplitude and phase response of the channel are different. In this case the waveform of the received signal would be different from that of the original transmitted signal. Intersymbol interference may also result from a channel multipath effect [115], in which the transmitted signal reaches the destination through different paths. As a result with different time lags the aggregated waveform will be distorted.

The effect of intersymbol interfence on communication system performance can be demonstrated by the example of Pulse Amplitude Modulation (PAM) transmission. In PAM a synchronous modem transmitter collects an integral number of bits of data at a time and encodes them into symbols for transmission with amplitude of -1 or 1 at the signaling rate. At the receiver end intersymbol interfence makes each symbol extend beyond the time interval used to represent the symbol and overlap with adjacent symbols, or in another words the boundaries between symbols are blurred. As the correct detection of transmitted symbols depends on a clear distiction between -1 and 1 symbols at the receiver, the blur can lead to a high bit error rate. Thus some kind of compensative filtering is essential for a high performance transmission system. This filtering is usually called equalization, and the filter called an equalizer.

Since intersymbol interference is one of the major obstacles for high speed data transmission, it has been an active area of research, and many algorithms and filter structures have been considered for equalization [116]. There are two classes of equalization strategies. One class does not need any training signal and is called blind equalization [117]. In blind equalization the transmitted signal must have some features, like independent and identifical distributions for example, which can be exploited as a clue for equalization. In this chapter, we consider the second class of equalization in which a training signal is used. In this type of equalization, a prearranged signal known to the receiver can be transmitted at the begining as a training signal to establish a communication channel. One of the most widely used equalizers is the linear transversal equalizer. It has been shown that this kind of structure is not satisfactory for non-minimum phase channel compensation and multi-layer perceptron (MLP) neural networks have been proposed as possible structures for equalizers [118]. In this paper we consider the use of Kohonen self-organization maps as an alternative structure for adaptive equalizers.

The intersymbol interference effect can be modeled by a finite impulse response (FIR) filter. Actually this is the most commonly adopted model. Based on this model, an equalization system may be represented as in Fig-6.1. The input signal sequence x_i is composed of transmitted symbols with amplitudes of -1 and 1. The transmission channel is modeled by an FIR filter with real coefficients a_i $(i = 1, \cdots, n)$, which is used to model the intersymbol interference effect. Its **Z** transform is $a_0 + a_1 z^{-1} + \cdots + a_n z^{-n}$. The output y_i is represented as

$$y_i = a_0 x_i + a_1 x_{i-1} + \cdots + a_n x_{i-n} = \sum_{k=0}^{n} a_k x_{i-k}$$

n_i is the additive noise of the channel, which has a zero mean normal distribution. The noise distorted y_i is represented as $\hat{y}_i$. The function of the equalizer is to use $\hat{y}_i$, $\hat{y}_{i-1}$, $\cdots$, $\hat{y}_{i-m+1}$ as input and to produce the best estimation of x_i (or

x_{i-d} in the delayed equalization cases), where m is the order of the equalizer. In the following sections, we consider in more detail different kinds of structures for equalizers.

6.2 Minimum Phase Channel and Equalizers

One kind of equalizer is the linear transversal equalizer. It can be described as

$$\hat{x}_i = sgn(\mathbf{B}^T \mathbf{Y}_i)$$

where $\mathbf{Y}_i$ denotes the vector of observed channel outputs $(\hat{y}_i,\ \hat{y}_{i-1},\ \cdots,\ \hat{y}_{i-m+1})$, $\mathbf{B}$ is the coefficient vector of the equalizer which is $(b_0,\ b_1,\ \cdots,\ b_{m-1})$, and sgn(x) is defined as

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

As the characteristics of the channels are not known a priori, the coefficients b_i are usually obtained by using adaptive algorithms, for example the LMS algorithm.

If a linear transversal equalizer is allowed to have infinitely high order, that is the coefficient vector can have unlimited length, then theoretically, all intersymbol interference can be compenseted at the output end. Zero intersymbol interference can be reached, because an infinite-length equalizer can implement a filter which has the exact inverse frequency response to that of the channel. However as it is impractical to implement an infinite-length filter, in practice only finite-length filters are used to approximate the ideal filter. Under this condition, whether the linear transversal equalizer can correctly recover the symbol x_i depends on the channel model coefficients a_j $(j = 0,\ \cdots,\ n)$ [118]. This can be shown as follows. Let $P_m(1)$ be defined as

$$P_m(1) = \{\mathbf{Y}_i \in R^m \mid x_i = 1\}$$

where $\mathbf{Y}_i$ is defined before. $P_m(-1)$ can be defined in similar way. Thus $P_m(1)$ and $P_m(-1)$ represent the sets of possible channel output vectors $(y_i, \cdots, y_{i-m+1})$ which can be produced from sequences of channel inputs beginning with $x_i = 1$ and $x_i = -1$ respectively. If we assume that additive noise is absent, that is $\hat{y}_i = y_i$, then from the foregoing description of the linear transversal equalizer, it is clear that x_i can be correctly recovered if and only if $P_m(1)$ and $P_m(-1)$ are linearly separable for some integer m. This condition is related to the channel model coefficients a_j $(j = 0, \cdots, n)$ by the following two theorems.

Theorem6.1: The condition for $P_m(1)$ and $P_m(-1)$ to be linearly separable is that there exist a sequence $(b_0, b_1, \cdots, b_{m-1})$ which satisfies

$$C = A \otimes B \quad and \quad c_0 > \sum_{k=1}^{m+n-1} |c_k| \qquad (6-1)$$

where A is the sequence $(a_0, a_1, \cdots, a_n)$ and B is the sequence $(b_0, b_1, \cdots, b_{m-1})$, $\otimes$ represents convolution. (For proof see Appendix)

Theorem6.2: For the sequence A, the necessary and sufficient condition for the existence of a sequence B $(b_0, b_1, \cdots, b_{m-1})$ which satisfies

$$C = A \otimes B \qquad c_0 > \sum_{k=1}^{n+m-1} |c_k|$$

is that the polynomial

$$A: \quad a_0 z^n + a_1 z^{n-1} + \cdots + a_n$$

has all its roots lie strictly within the unit circle in the complex plane. (For proof see Appendix)

From the the proof of the theorems given in the Appendix, it may be concluded that if the roots of the A polynomial lie closer to the unit circle, then m should be larger. That means a high order linear transversal equalizer should be used.

Now it has been proved that only under the minimum phase channel (equivalent to the condition that all roots of the A polynomial lie strictly in the unit circle) can the linear transversal equalizer correctly recover all the symbols input to the channel at any time if the additive noise is absent. For non-minimum phase channels, to use a linear transversal equalizer, some delay must be introduced into the equalizer. Thus the output of the equalizer is $\hat{x}_{i-d}$ which is the estimation of x_{i-d} rather than x_i. Fig-6.2a and Fig-6.2b show the distribution of a minimum phase channel $P_m(1)$ and $P_m(-1)$ and a non-minimum phase channel $P_m(1)$ and $P_m(-1)$ respectively. Fig-6.2c shows the distribution of non-minimum phase channel $P_m(1)$ and $P_m(-1)$ with additive noise. Thus to use a linear transversal equalizer for channel equalization, the channel should be minimum phase or some delay should be used.

The above characteristics of linear transversal equalizers make them unsuitable for nonstationary channel equalization. In this case, the minimum phase condition cannot be guaranteed and the delay needed is also varying. From the foregoing discussion it can be seen that equalization may be regarded as a pattern classification problem, input vectors are classified into $P_m(1)$ and $P_m(-1)$ classes. Thus MLP neural networks have been considered as a structure for equalizers [118], and it is also has been implemented with hardware and extended to the Decision Feedback Equalizer [119][120]. As MLP neural networks can realize any continuous mapping, linear separability is no longer an obstacle for MLP equalizers. However since the MLP neural networks have the local minimum problem, they may give a suboptimal division of $P_m(1)$ and $P_m(-1)$, or in the worst case an incorrect division. From the distribution pattern of $P_m(1)$ and $P_m(-1)$ shown in Fig-6.2c, it can be seen that clustering algorithms may be more suitable for classifying the sample points into $P_m(1)$ and $P_m(-1)$ classes rather than a piecewise linear dividing algorithm, like the

MLP. As the Kohonen self-organizing feature map is very similar to the K-means clustering algorithm [6], it can be used as a structure for the equalizer.

6.3 The Kohonen Self-organizing Feature Map as a Structure for Channel Equalization

The Kohonen self-organizing feature map reflects the organizing principle of biological neural systems. In the brain there are a lot of fine-structures and many of them are determined genetically. However there also exists direct experimental evidence that some of these structures are formed by a self-organizing process which depends on experience [121]. The Kohonen self-organizing feature map was introduced by Kohonen as a model of biological self-organizing processes.

The structure of a Kohonen self-organizing feature map is shown in Fig-6.3. Each unit in the map is connected to n input units, n is the dimension of the input vectors. Continuous-valued input vectors are presented sequentially in time without specifying the desired output. After enough input vectors have been presented, weights will specify cluster or vector centers that sample the input space such that the point density function of the vector centers tends to approximate the probability density function of the input vectors [6]. This feature can be an advantage for classifying the distribution patterns such as shown in Fig-6.2c.

The self-organizing algorithm can be described by the following steps

1: *Initialize weights which connect the units in the map and the input units with small random values. Set the initial radius of the neighborhood.*

2: *Present new input vector.*

3: *Compute distances d_j between the input vector and each unit j in the map using*

$$d_j = \sum_{i=1}^{n} (x_i(t) - m_{ij}(t))^2$$

where $x_i(t)$ is the value of input units i at time t, and $m_{ij}(t)$ is the weight from input unit i to the unit j inthe map at time t.

4: *Select the unit k which has the minimum distance value d_{min} in the map.*

5: *Weights are updated for the unit k and all the units in the neighbourhood defined by $N_k(t)$. The update formula is*

$$m_{ij}(t+1) = m_{ij}(t) + \alpha(t)(x_i(t) - m_{ij}(t)) \quad (i = 1, \cdots, n)$$

The term $\alpha(t)$ is the stepsize which in a similar manner to the neighbourhood decreases in time .

6: *Repeat by going to step 2.*

A self-organizing feature map formed by using this algorithm and the samples in Fig-6.2c as input vectors is shown in Fig-6.4. Its advantage for classification is very clear.

The self-organizing map described above can be easily transformed into an adaptive equalizer. For the purpose of equalization, the map is split into two submaps in the middle. The left part is for the input vectors from the $P_m(1)$ set, and the right part for the input vectors from the $P_m(-1)$ set. Thus when an input vector is presented to the input units, if it belongs to $P_m(1)$, then the weights in the left of the map will be updated using the algorithm described above, otherwise the weights of the right part of the map will be updated. On top of the map

there is a decision unit whose function can be described as

$$output = \begin{cases} 1, & \text{if } \sum w_j y_j > 0; \\ -1, & \text{otherwise.} \end{cases}$$

where y_j is the output state of unit j in the map, which is 1 if its weight vector is the closest to the input vector, otherwise it is 0. This can be implemented by lateral inhibitation neural networks like MAXNET [6], or by software. The parameter w_j is the weight which connects the unit j in the map and the decision unit, it is 1 if unit j is in the left half of the map, and is -1 otherwise.

Fig-6.5 shows the bit error rate performance of an MLP equalizer and the self-organizing map equalizer. The MLP equalizer used a 5-9-3-1 structure which is described in [118], and the self-organizing map equalizer used a second order input, that is the input vectors are two dimensional. The channel model is $0.3482 + 0.8704z^{-1} + 0.3482z^{-2}$ which is the same as that in [118]. It is a non-minimum phase channel, so linear transversal equalizers cannot recover the original input symbols without some delay. It is clear from Fig-6.5 that the self-organizing map equalizer has a lower bit error rate than the MLP equalizer.

To compare with a linear transversal equalizer, a delay of one sample was introduced in the estimation of the channel input symbol. That is we are estimating x_{i-1} rather than x_i at time i. The self-organizing equalizer used fourth order input vectors in this case. The bit error rates are shown in Fig-6.6. The lowest curve is obtained by the self-organizing map equalizer. Both the curves for the MLP and linear transversal equalizer are as reported previously[118]. The improved performance obtained from the self-organizing equalizer is clear.

6.4 Conclusions

The Kohonen self-organizing feature map has several advantages as a structure

for channel equalization. Compared with linear transversal equalizers, it is not limited by the minimum phase channel condition. It is more robust in performance than MLP equalizers and does not suffer from a potential local minimum problem. Another advantage of the self-organizing equalizer is that its map feature is more suitable for classifing clustering distribution pattern of $P_m(1)$ and $P_m(-1)$ samples, so it can obtain a lower bit error rate. Finally it should be mentioned that the highly parallel structure of the Kohonen self-organizing algorithm would be ameanable to VLSI implementation.

In next chapter we discusss application of learning algorithms to ATM call access control in broadband ISDN.

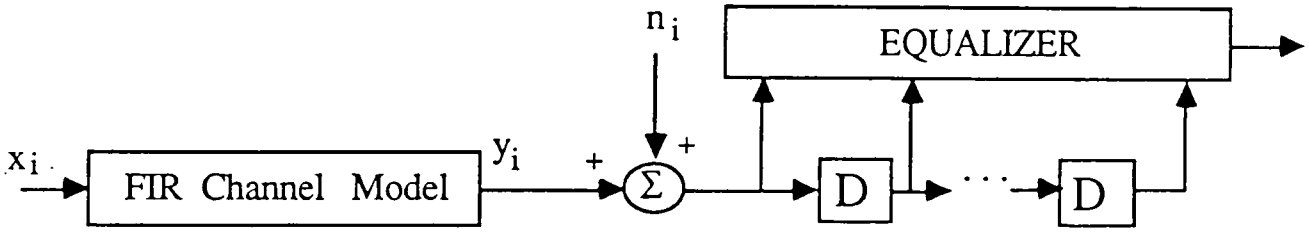
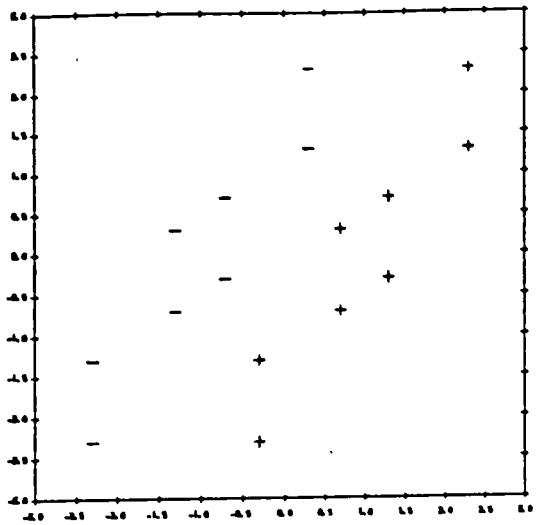
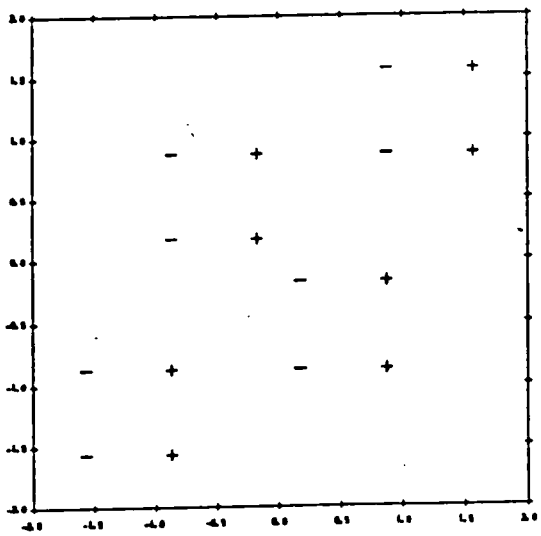


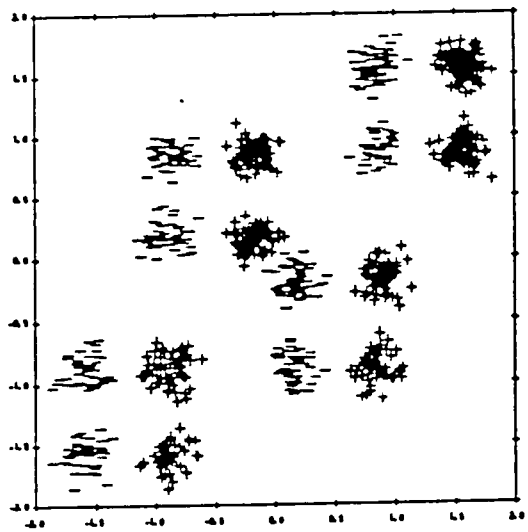
Fig-6.1. Schematic representation of channel model.



(a)



(b)



(c)

Fig-6.2. The distribution pattern of $P_m(1)$ and $P_m(-1)$ sets. (a) minimum phase channel. (b) non-minimum phase channel. (c) is the same as (b) except that additive noise is added.

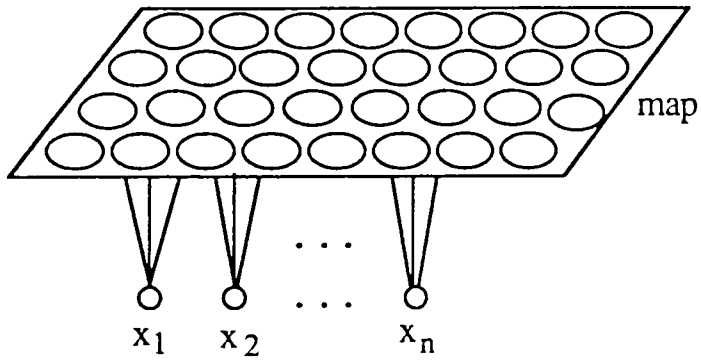


Fig-6.3. Structure of Kohonen self-organizing feature map.

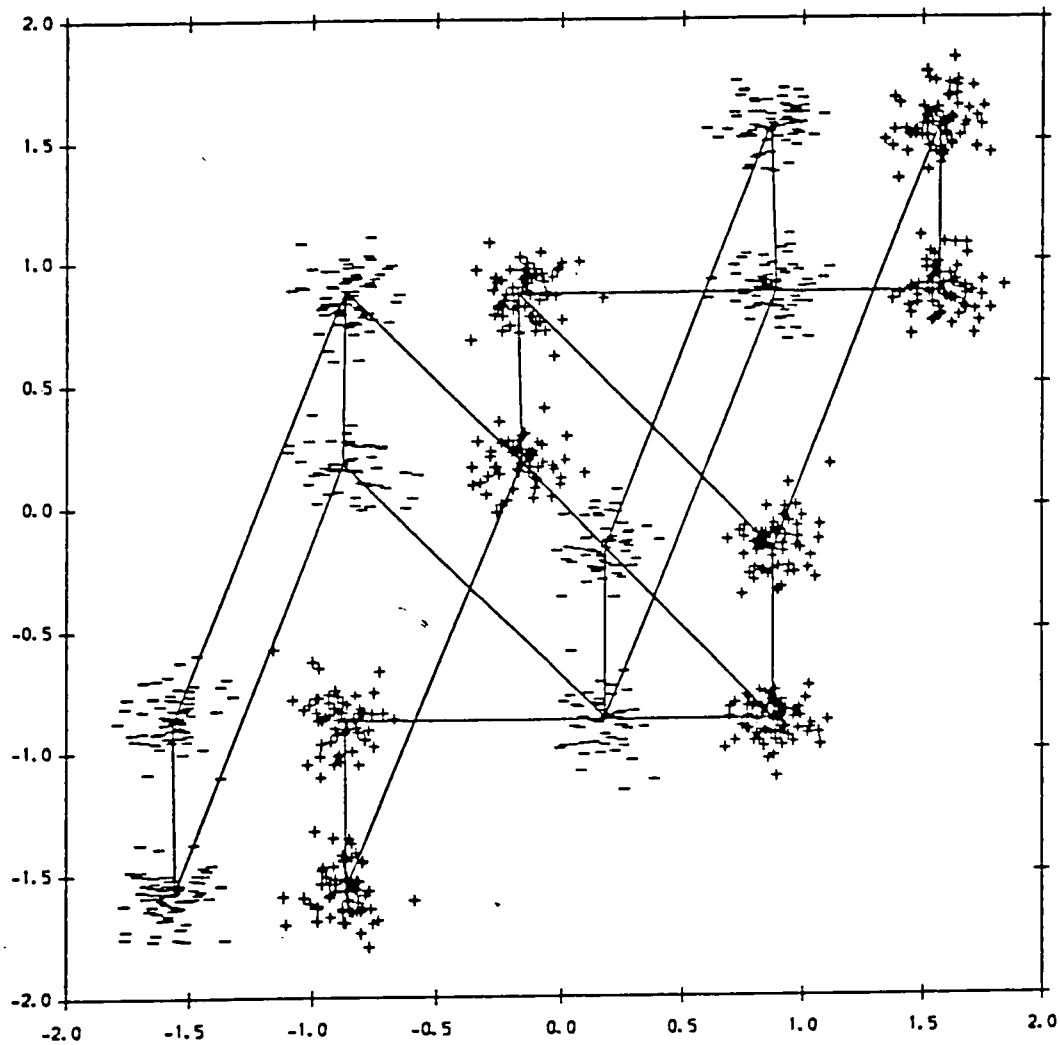


Fig-6.4. Feature map formed by the Kohonen self-organizing algorithm.

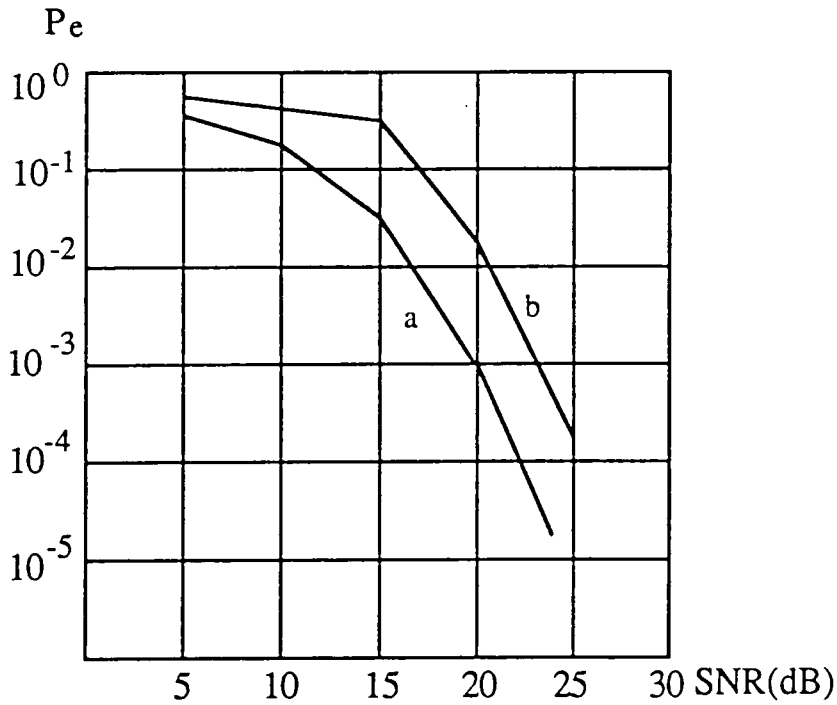


Fig-6.5. Bit error rate as a function of signal to noise ratio (SNR). (a) self-organizing equalizer. (b) MLP equalizer.

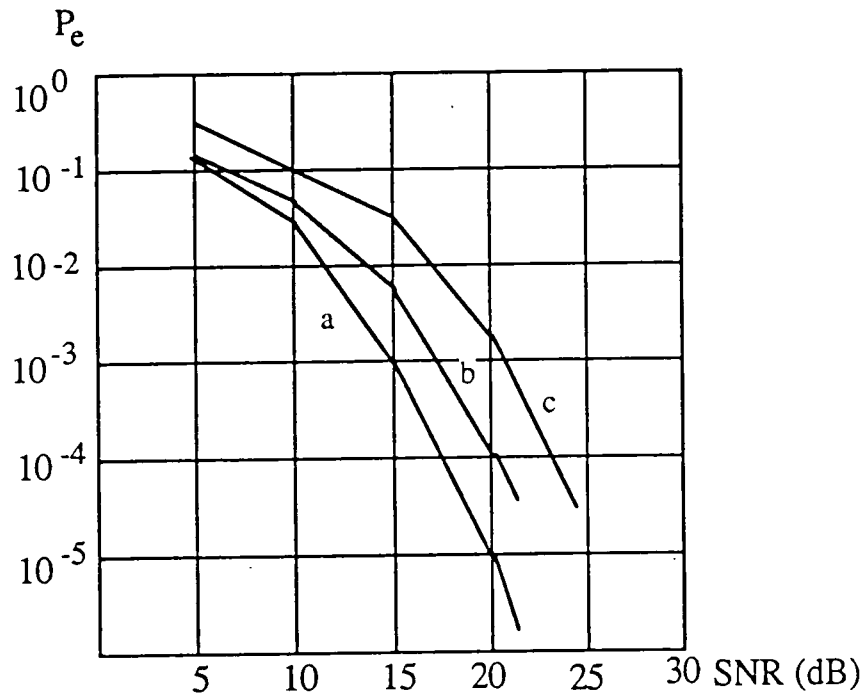


Fig-6.6. Bit error rate as a function of signal to noise ratio (SNR). (a) self-organizing equalizer. (b) and (c) are as reported in [118] using an MLP equalizer and linear transversal equalizer respectively.

Chapter Seven

Adaptive ATM Call Access Control Using Learning Networks

7.1 Introduction

The Broadband Integrated Services Digital Network (B-ISDN) is an emerging communication network which is intended to provide multimedia services to its customers in a flexible and cost-effective manner. The services include voice, video and data transmission. Research and development in B-ISDN is a very active area.

The traditional transport paradigm used for ISDN is synchronous transfer mode (STM) [122]. The rule for subdivision and allocation of bandwidth using STM is to allocate time slots within a recurring structure (frame) to a service for the duration of call. An STM channel is identified by the position of its time slots within a synchronous structure. The hierarchical channel structure of STM consists of several bearer channels, and each of them has different transmission rate. One of the drawbacks of appling STM to B-ISDN is its rigid bearer channel structure which make the dynamic allocation of time slots difficult [122]. In an B-ISDN environment, the services have greatly varied bit rate, and some kind of dynamic allocation of time slots (or bandwidth) is necessary to make efficient use of the bandwidth resource. Thus the asynchronous transfer mode (ATM) has attracted significant attention as a transport paradigm for B-ISDN [123] [124]. In ATM, specific periodic time slots are not assigned to a fixed service, useable bandwidth is segmented into fixed size information bearing units called packets or cells. Each cell consists of a header and an information field. The header contains a logical address, which identifies the virtual circuit to which the call is assigned, priority

information and a error detecting and correcting code. Data to be transmitted is conveyed in the information field. These cells can be dynamically allocated to services on demand. In comparison to STM, ATM is more flexible, and may have potential gain in bandwidth efficiency by buffering and statistically multiplexing bursty traffic at the expense of cell delay and loss [125]. To guarantee the quality of the services provided by the network, the cell loss rate and delay must be controlled within tolerable range by an appropriate network controller. In this chapter we concentrate on the statistical multiplexing control strategy and consider two access control strategies based on learning algorithms. We first discuss the basic problem of bandwidth resource manegement in ATM. Two new adaptive strategies are then considered with associated simulation results and a critical discussion.

7.2 The Call Access Control of ATM

ATM has a layered function structure which is shown in Fig-7.1. The ATM adaption layer transforms the information stream originated from a user terminal or end system into fixed length cells according to the ATM format. These cells are buffered and asynchronously multiplexed and/or switched by the ATM transport layer. All these functions are supported by the electronic circuits and transmission link in the physical layer.

To guarantee performance requirements like cell delay and loss demanded by the services which are supported by the B-ISDN, a call access control strategy (or traffic control strategy, call regulation) must be implemented in the transport layer to control the quality of the services. When an ATM terminal initiates a call request to the network, the network manager must then check that there is sufficient bandwidth resource to provide the connection requested with satisfactory quality of service, or the request is rejected. Generally, there are two call regulation rules

[125]. One is nonstatistical multiplexing, in which if the sum of the peak cell rate of all the hold on calls (including the new incoming call) does not exceed the output link rate, then the new incoming call is accepted. Otherwise it would be rejected. That is the call accept condition is

$$\sum_i P_i \leq C$$

where P_i is the peak rate of the ith hold on call, and C is the capacity of the output link at the node. This approach is quite similar to bandwidth reservation for STM, but with the added flexibility of being able to reserve any peak rate required, rather than a multiple of a base channel rate. The strong advantage with nonstatistical multiplexing is minimal cell delay and no cell loss due to buffer overflow. However in the case when a large proportion of the traffic flow in the link is bursty, nonstatistical multiplexing can show low efficiency in making use of bandwidth resource. Thus statistical multiplexing is considered to exploit the burstiness of traffic flow and obtain potential gain in bandwidth efficiency. In statistical multiplexing, the total peak cell transmission rate of all the accepted calls is allowed to exceed the capacity of the link at the expense of cell delay or cell loss. However under a proper control strategy the cell delay or cell loss can be controlled within a tolerable range.

Statistical multiplexing can only increase bandwidth efficiency under certain conditions. The preconditions are that the average burst length B of calls is short, the peak rate to link capacity ratio (PLR) of calls is low and the burstiness of calls are high [125]. Let P denote the peak rate of a call, A the average rate and C the capacity of the link, then the burstiness of the call is defined as P/A, and $PLR = P/C$. In [125] the authors give some computer simulation results on the feasibility of using statistical multiplexing in homogeneous traffic and heterogeneous traffic enviroments. In the homogeneous traffic case, generally PLR should be less than 0.1. These preconditions can be met in many cases in B-ISDN due to wide bandwidth

and inherently bursty data services. Also advanced image coding techniques are making the traditional continuous sources like video into bursty sources [126]. To obtain an approximate estimation of the possible bandwidth usage efficiency gain, consider an analysis based on a simple homogeneous traffic model. In this model all the incoming calls have the same burstiness and the average cell rate is 0.5P, where P is the peak cell rate. We assume that N calls have been accepted. If the number of accepted calls are large, then the total cell rate can be approximated by a normal distribution, as each call is an independent source emitting cells at rate between 0 and P randomly. The normal distribution can be denoted as G(M, σ^2), where

$$M = N \times 0.5P = \frac{NP}{2}$$

To estimate the variance σ^2, we assume the variance for each call is $0.25P^2$. This is a conservative estimation. A random variable with a value between 0 and P, has $0.25P^2$ as maximum variance. Thus we have

$$\sigma^2 = n \times 0.25P^2 = \frac{NP^2}{4} = \frac{1}{N}(\frac{NP}{2})^2 = \frac{M^2}{N}$$

Thus the call rate in the link can be approximated by the normal distribution G(M, $\frac{M^2}{N}$). If we want to keep the overload probability below p_{ov} (or cell loss rate below specification), then we must have

$$\frac{C - M}{\sigma} \geq B \qquad (7-1)$$

where C is the capacity of the link and B is given as

$$\int_B^{+\infty} \frac{1}{\sqrt{2\pi}} exp(-\frac{x^2}{2}) dx = p_{ov} \qquad (7-2)$$

Here it is implicitly assumed that $C > M$. As M represents the mean cell rate in the link, it is a practical assumption. From inequality (7-1) we have

$$C - M \geq B\sigma$$

As σ^2 can be represented by M and N, it is obvious that

$$C - M \leq B\frac{M}{\sqrt{N}} \tag{7-3}$$

If both sides of (7-3) are divided by C, then

$$1 - \frac{M}{C} \geq \frac{B}{\sqrt{N}}\frac{M}{C}$$

that is

$$\frac{M}{C} \leq \frac{1}{1 + \frac{B}{\sqrt{N}}} \tag{7-4}$$

The value of $\frac{M}{C}$ can be regarded as a measure of the efficiency of the bandwidth usage. Then (7-2) and (7-4) can be used to estimate the possible efficiency gain at different p_{ov} and N values. Table-7.1 gives the efficiency estimation for several p_{ov} and N values.

Overload Probability	N value	Estimated Efficiency
0.0001	50	65.53%
	100	72.83%
	1000	89.47%
0.001	50	69.58%
	100	76.39%
	1000	91.09%
0.01	50	75.21%
	100	81.10%
	1000	93.14%

Table-7.1

From Table-7.1 it can be seen that for large N, there can be significant gain in bandwidth efficiency by using statistical multiplexing. As an example with $p_{ov} = 0.0001$ and $N = 1000$ using proper statistical multiplexing, the bandwidth efficiency

can be around 89%. While using STM or nonstatistical multiplexing the efficiency is only 50%, as the ratio of mean cell rate to peak cell rate is 0.5 for all the incoming calls in this case. The potential gain is significant. However as mentioned in the foregoing discussion $C > M$, that is the capacity of the link is larger than the sum of average cell rate of all the calls in the link. This restriction puts an upper bound on the value of N. To have large N, the PLR should be small. In the above case, the peak cell rate of incoming calls should be around one thousandth of the capacity of the link. In the following parts of the report we usually assume low PLR value and high burstiness of services.

The services supported by B-ISDN may vary from narrowband to wideband and from continous to bursty, and their performance requirements are also different. For example, image and video services require low cell loss rate, interactive services like telephony requires short cell delay, and some services like data file transfer may have less stringent requirement on cell delay and loss. However to implement a call regulation strategy which can meet diverse performance requirements in B-ISDN would make the cost-effectivness of B-ISDN questionable. One simple strategy is to use the most stringent performance requirement which is adequate for all services supported by B-ISDN [125]. More sophisticated control strategies may divide services into a few classes, each class has its own performance requirement and may be delay sensitive or cell loss sensitive [127] [128]. It is a trade-off problem to decide how the call regulation strategy will cover the different performance requirements. In the following discussion we assume only one class of performance requirement.

The traffic control strategy of traditional communication networks like ISDN is based on a through study of the statistical traffic characteristics in the network. In the case of B-ISDN the diverse variety of services and topology or connection routing evolution make this kind of study very difficult. Thus it is desirable to have

the call regulation rule of ATM nodes node architecture independent and robust to traffic uncertainties [125]. This suggests that some kind of adaptive or learning call regulation would be highly desirable. In the next section, we discuss two adaptive call regulation strategies.

7.3 Adaptive Call Access Control Strategies

Adaptive call regulation can use learning automata [129] or an artificial neural network [130] as its basic structure. The neural network controller considered in [130] used the back-propagation algorithm for training. Thus it has a potential local minima problem and retraining is not easy. Another undesirable feature of this controller is that it only uses the incoming cell pattern in the link as the basis for call regulation and does not take into account the fact that different type calls may require different bandwidth. Sometimes the traffic condition on a link may not be able to support a wideband service but is adequate for a narrowband call. This kind of situation cannot be dealt with efficiently by the neural network controller mentioned above. If the traffic characteristics can be approximated by a normal distribution, a simple linear call regulation rule may be used. In the following we discuss a perceptron like adaptive call regulation controller which uses a linear inequality as decision rule.

The ATM node model is depicted in Fig-7.2. It has many input ports and the incoming cells are statistically multiplexed and transmitted through the output link.

A general call source model is considered as a Markov chain [125]. A first-order Markov source is used in the following discussion which is depicted in Fig-7.3. During the active period the source emits cells at its peak rate, and in passive period

no cells are emitted. For this model the average burst length B can be calculated as

$$B = b + 2(1-b)b + 3(1-b)^2 b + 4(1-b)^3 b + \cdots\cdots$$
$$= \frac{b}{(1-1+b)^2} = \frac{1}{b}$$

and the average cell rate is

$$A = \frac{aP}{a+b}$$

where P is the peak rate of the source.

A special case for the above model is when $a + b = 1$ and the source degrades into an uncorrelated source, and the probability distribution of cell rate at every moment is a Bernoulli distribution.

As mentioned previously if the PLR value is small, the number of calls which can be transmitted simultaneously through the link can be very large. If these calls are statistical independent and are uncorrelated sources which are described above, then according to the central limit theorem [131], the statistical characteristics of the traffic mix in the link can be approximated by a normal distribution. Assume there are three classes of hold on calls and the number of calls are N_1, N_2 and N_3. The probability distribution of each call's cell rate is a Bernoulli distribution with mean of m_1, m_2 and m_3 and variance σ_1, σ_2 and σ_3. The probability distribution of mixed cell rate can be approximated by a normal distribution N(m, σ), where

$$m = N_1 m_1 + N_2 m_2 + N_3 m_3$$

and

$$\sigma = \sqrt{N_1\sigma_1^2 + N_2\sigma_2^2 + N_3\sigma_3^2}$$

Fig-7.4 shows a simulation result on the distribution of cell rate in a mixed traffic link. It is very close to a normal distribution. If we assume the link capacity is C,

then the overflow probability is

$$p_{ov} = \int_C^{\infty} \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(x-m)^2}{2\sigma^2})dx$$
$$= \int_{\frac{C-m}{\sigma}}^{\infty} \frac{1}{\sqrt{2\pi}} exp(-\frac{y^2}{2})dy$$

The condition for p_{ov} to be less than a specific value can expressed as

$$\frac{C-m}{\sigma} > K \qquad (7-5)$$

where K is a constant. The expression (7-5) can be rewritten as

$$K\sigma < C - m \qquad (7-6)$$

If we square both sides of (7-6) we obtain

$$K^2\sigma_1^2 N_1 + K^2\sigma_2^2 N_2 + K^2\sigma_3^2 N_3 < C - 2Cm_1N_1 - 2Cm_2N_2 - 2Cm_3N_3$$
$$+ (m_1N_1 + m_2N_2 + m_3N_3)^2 \qquad (7-7)$$

from the small PLR assumption, it is clear that the coefficients of second order terms like m_i^2 and m_im_j are much smaller than $2Cm_i$, so the second order terms can be ignored as an approximation. Then we obtain

$$(2Cm_1 + K^2\sigma_1^2)N_1 + (2Cm_2 + K^2\sigma_2^2)N_2 + (2Cm_3 + K^2\sigma_3^2)N_3 < C \qquad (7-8)$$

Thus a simple linear inequality can be used to control p_{ov}. As the cell loss rate r_{ls} is a monotonic function of p_{ov} a linear control strategy can be used to control r_{ls}. The simulation results presented in [132] and [133] also suggested a linear control strategy, but here we give a more rigorous treatment and suggest a method for calculating the control coefficients.

7.3.1 Perceptron Control Rule

From the above discussion, we can suggest an adaptive call regulation rule based on the inequality

$$a_1 N_1 + a_2 N_2 + a_3 N_3 < T$$

where N_1 is the number of hold on calls of narrowband calls, and N_2 and N_3 are for intermediate-band and wideband calls. Each class of call has different cell rates. Of course one can classify calls into more classes, but for the simplicity of control rule, large number of classes are unfavorable. The coefficients a_1, a_2 and a_3 can be adaptively updated using a perceptron like algorithm [16] as follows.

If a call request is accepted but the following cell loss rate exceeds the performance requirement, then

$$a_1(n+1) = a_1(n) + \alpha N_1 \quad a_2(n+1) = a_2(n) + \alpha N_2 \quad a_3(n+1) = a_3(n) + \alpha N_3$$

where α is the learning stepsize. If the call is rejected and the cell loss rate is much lower than the performance requirement, then

$$a_1(n+1) = a_1(n) - \beta N_1 \quad a_2(n+1) = a_2(n) - \beta N_2 \quad a_3(n+1) = a_3(n) - \beta N_3$$

where β is the learning stepsize.

Although the above discussion is based on an uncorrelated source model assumption, for the correlated Markov source model with $a + b \neq 1$ the normal distribution approximation is still valid. Fig-7.5 shows the simulation result on distribution of cell rate in a link containing mixed calls with first-order Markov model sources. The distribution is again very close to a normal distribution. Only the functional relation between p_{ov} and r_{ls} will be changed, but it is still a monotonic

function. Thus the linear inequality control rule is still applicable. The simulation results presented in section 4 show that it can obviously improve the efficiency performance.

7.3.2 RAM Map Control Rule

Both the perceptron mentioned above and the neural network controller considered in [130] essentially implement a functional mapping. If we consider that the output of the controller is just accept or reject and the input can usually be transformed into binary form, these mappings are just Boolean logic functions. and can be implemented using digital logic circuits If variables N_1, N_2 and N_3 are viewed as three orthogonal axes, then every combination of hold on call pattern in a link is represented as an integer point in the space. All these points form a lattice array with finite nodes. This lattice array can be implemented by a RAM which is depicted in Fig-7.6. N_1, N_2 and N_3 are represented as binary numbers to the address lines, the output is single bit with 1 representing accept and 0 for reject. For a RAM with 16 address lines, it contains a $2^{16} = 65536$ node lattice, and is sophisticated enough for most applications. To train this RAM network, a learning paradigm which is similar to the self-organization map algorithm [121] is introduced. The learning algorithm can be explained with the help of Fig-7.7. To simplify the discussion, we assume there are only two classes of calls. N_1 axis represents the number of calls which belong to class one, and the N_2 axis for the calls of class two. When there are n_1 class one calls and n_2 class two calls in the link, then the system state can be represented by a point P which is shown in Fig-7.7. If the cell loss rate exceeds the performance requirement, then the nodes in the upper-right neighbourhood of P (which is shown as a shadowed square) including the nodes on the boundary will be assigned the value of 0. When the system state come into this region later, new incoming calls will be rejected. If a call is rejected at point P, and the cell loss rate

is much lower than the performance requirement, then the nodes in the lower-left neighbourhood of P will be assigned the value of 1. When the system state is in this region, a new incoming call will be accepted. The neighbourhood used in this case is a square, and in high dimensional cases, it would be super cubic. Its size decreases during the training process to reduce random fluctuations. Utimately, the learning algorithm will divide the map into an A-region (Accept region) and an R-region (Reject region) as shown in Fig-7.8.

One advantage of this approach is that it can implement any possible nonlinear mapping. This is in contrast to the perceptron which is limited to a linear decision rule. Secondly, compared with the MLP neural network trained with a Back Propagation Algorithm, it is easier to retrain because it has no local minima problem. In the case of a biased distributed learning sample set, the RAM self-organization map learning does not need to use a leaky pattern table (LPT) method which is essential for biased learning in an MLP neural network [130]. This can reduce the computation time significantly. The drawback is its generalization ability. For a perceptron or MLP neural network each learning sample moves the whole decision line to a new position, while in the RAM self-organization map learning each learning sample can only change a small local portion of the whole decision boundary which lies in its neighbourhood. Thus the RAM self-organization technique has a smaller generalization range.

7.4 Simulation Results and Discussion

A flow chart of the simulation program is shown in Fig-7.9 and Fig-7.10. In module-B, a Poisson flow source is simulated. This is achieved by using a random number generator whose distribution function is approximately an exponential function to generate the time interval between incoming calls. The simulated Poisson

call flow is later demultiplexed into three subflows to simulate three different classes of calls which have different peak cell rate and burstiness. Module-C simulates the call access control rule. It can be a neural network or any other control strategies. In module-F, a learning algorithm is implemented.

To simulate a discrete time system, the sampling period or basic time unit of the simulation needs to be established. In the simulation reported here, the basic time unit for the interval between incoming calls is one second, and the basic time unit for traffic condition monitoring is 10ms. If the cell size is 50 bits, then the capacity of the link simulated is 500kb/s. It is clearly lower than that of the future ATM network. However the simulation is at cell level and to keep the simulation time within a reasonable range, it is a practical assumption. For the three classes of calls, narrowband has the rate of 5kb/s, mediumband 10kb/s and wideband 50kb/s. As the purpose of the simulation is to investigate the potential bandwidth efficiency gain from statistical multiplexing of calls in an ATM network, the intensity of incoming call flow is assumed larger than the capacity of the link.

For the perceptron control rule discussed in section 3, the $\alpha - LMS$ learning algorithm [10] is used for training. That is the stepsize which is defined in section 3 decreases with time. This helps to reduce the random fluctuation which is unavoidable in learning in a stochastic enviroment. Fig-7.11 and Fig-7.12 show the learning curves of $a_1(n)$ with decreasing stepsize and constant stepsize respectively. The curve in Fig-7.11 is smoother. The convergence properties of the LMS learning algorithm are fully discussed in [134]. Fig-7.13 and Fig-7.14 show the learning curves of coefficients $a_2(n)$ and $a_3(n)$ respectively. Both curves shows a clear convergence.

To verify the analytical prediction, the homogeneous traffic situation was simulated, and the simulation results are shown below. They are obtained with a

perceptron control rule.

N (Average number of on hold calls)	p_{ov}	Efficiency	Theoretical Estimation
154	0.000389	73.9%	78.8%
75	0.000552	70.3%	72.8%
47	0.001952	69.9%	70.4%

Table-7.2

Using the value of N and p_{ov} and formula (7-2) and (7-4) in section 2, the analytical estimation of the efficiency are shown in the last column in Table-7.2. It is obvious that the theoretical predictions are close to the simulation results.

Control strategy	Efficiency	Average peak rate	p_{ov}	Cell loss rate
nonstatistical mutiplexing	41.4%	97	0	0
perceptron control rule	54.8%	115	0.0018	7.57×10^{-5}
perceptron control rule	59.8%	121	0.0005	1.75×10^{-5}
RAM map control rule	57.7%	116	0.0016	5.25×10^{-5}
RAM map control rule	62.1%	123	0.0050	1.68×10^{-4}

Table-7.3

Control strategy	Efficiency	Average peak rate	p_{ov}	Cell loss rate
nonstatistical multiplexing	44.3%	97	0	0
perceptron control rule	61.7%	123	0.0008	3.85×10^{-5}
perceptron control rule	56.8%	122	0.00006	1.86×10^{-6}
RAM map control rule	67.6%	134	0.0062	2.12×10^{-4}
RAM map control rule	68.8%	138	0.0053	1.79×10^{-4}

Table-7.4

The simulation results for a heterogeneous traffic situation are shown in Table-7.3 and Table-7.4. The composition of incoming call flow is 60% narrowband calls,

30% mediumband calls and 10% wideband calls. Table-7.3 shows the simulation results for a short burst call source which has the average burst length 4.25 cells and the burstiness of 2. Table-7.4 shows the simulation results for long burst call source which has the average burst length 9.5 cell and burstiness of 2. There are two ways to measure the bandwith efficiency. One is to use the utilization of the capacity of the link like that defined in (7-4), and the other way is to measure the average peak cell rate or the actual throughput in the link. Both of these parameters are listed in the tables and the capacity of the link is normalized to 100.

It can be seen there is an obvious gain in bandwidth efficiency from statistical multiplexing (or statistical call access control) at the expense of some cell loss. Generally, the efficiency would increase with an in increase of p_{ov}or cell loss rate. However the results in Table-7.3 and Table-7.4 suggest that this may not be always the case. This can be explained by the variation of composition of calls in the link. In the simulation the composition of the incoming call flow is kept constant to evaluate the performance of the control rules. However the control rules which are hyper-surfaces or curves in the two dimensional case (see Fig-7.8) may have slightly different shape or position because of statistical learning. If the incoming call flow has an intensity greater than the capacity of the link, the system state would bounce around the decision surface. The variation of the surface cause the change in the composition of the accepted calls in the link. Different classes of calls have different peak cell rate and burst characteristics, and these features have a significant influence on the relationship between efficiency and p_{ov}. Thus the variation of composition of accepted calls can distort the monotonic relation between efficiency and p_{ov}.

Another feature shown in Table-7.3 and Table-7.4 is that the p_{ov} and the cell loss rate are always fluctuating. This is caused by learning in a stochastic enviro-

ment. A small learning stepsize or a learning neighbourhood can help to reduce the fluctuation, but the learning speed or the adaptation speed to a change in the enviroment would be decreased. Thus a compromise must be made between steady state accuracy and speed of convergence.

7.5 Conclusion

In comparison to STM, ATM has more flexibility in bandwidth resource mangement. To explore the potential gain in bandwidth usage efficiency which is made possible by ATM, statistical multiplexing is needed in call access control. Of course there are some preconditions which are necessary for profitable use of statistical multiplexing. The calls must have high burstiness and low PLR (peak cell rate to link capacity ratio). Under these conditions, statistical multiplexing with the learning algorithms discussed in this chapter can provide an obvious gain in bandwidth efficiency. This has been demonstrated by simulation results. Further work is needed on more intensive simulation studies and eventually testing on real ATM communication networks.

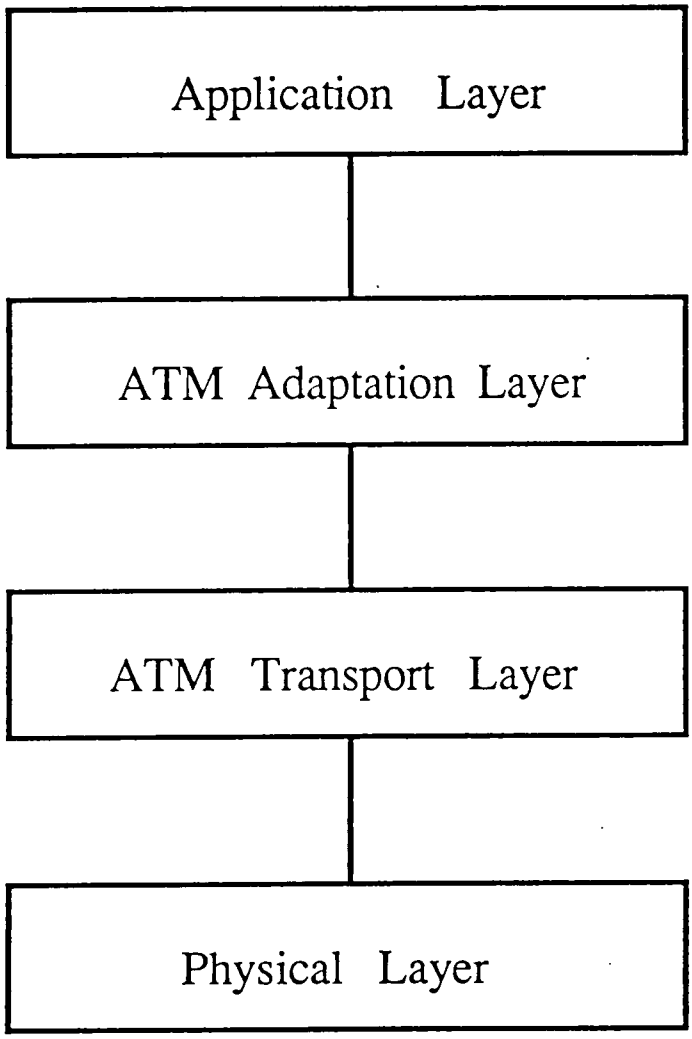


Fig-7.1. The layered structure of ATM

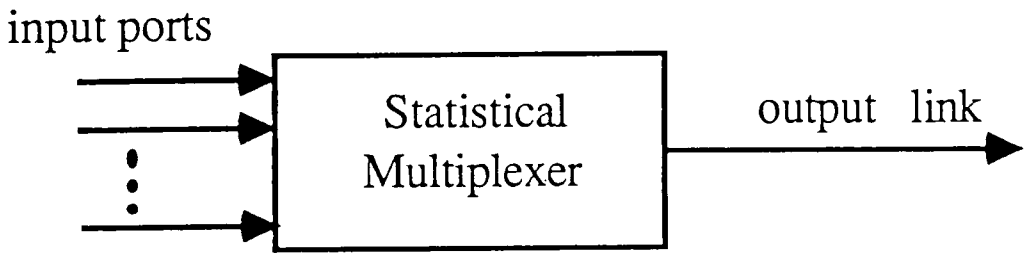


Fig-7.2. The ATM node model

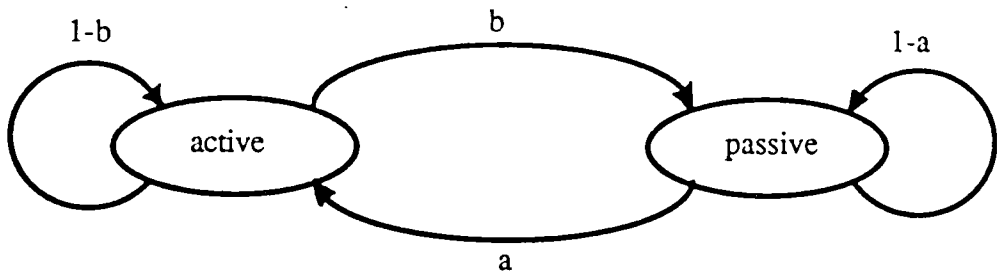


Fig-7.3. The ATM call source model

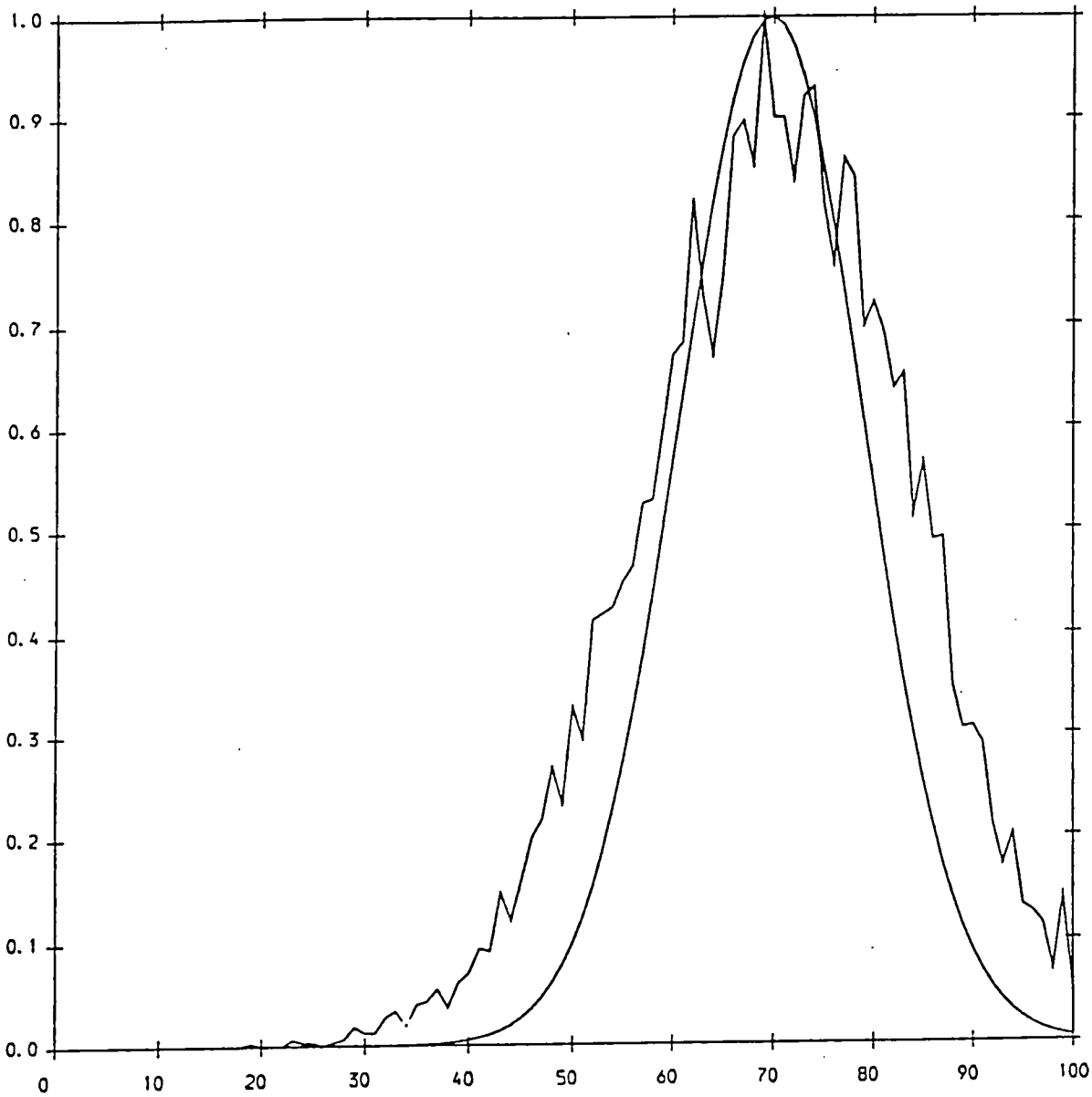


Fig-7.4. The cell rate distribution in the link contains mixed calls. $N_1 = 30$ $N_2 = 10$ and $N_3 = 5$. If we assume the capacity of the link is 100, then peak cell rate of class one calls $P_1 = 1$, and $P_2 = 5$, $P_3 = 10$. The burstiness are 2, 1.67 and 3.33 respectively. The distribution is close to a normal distribution.

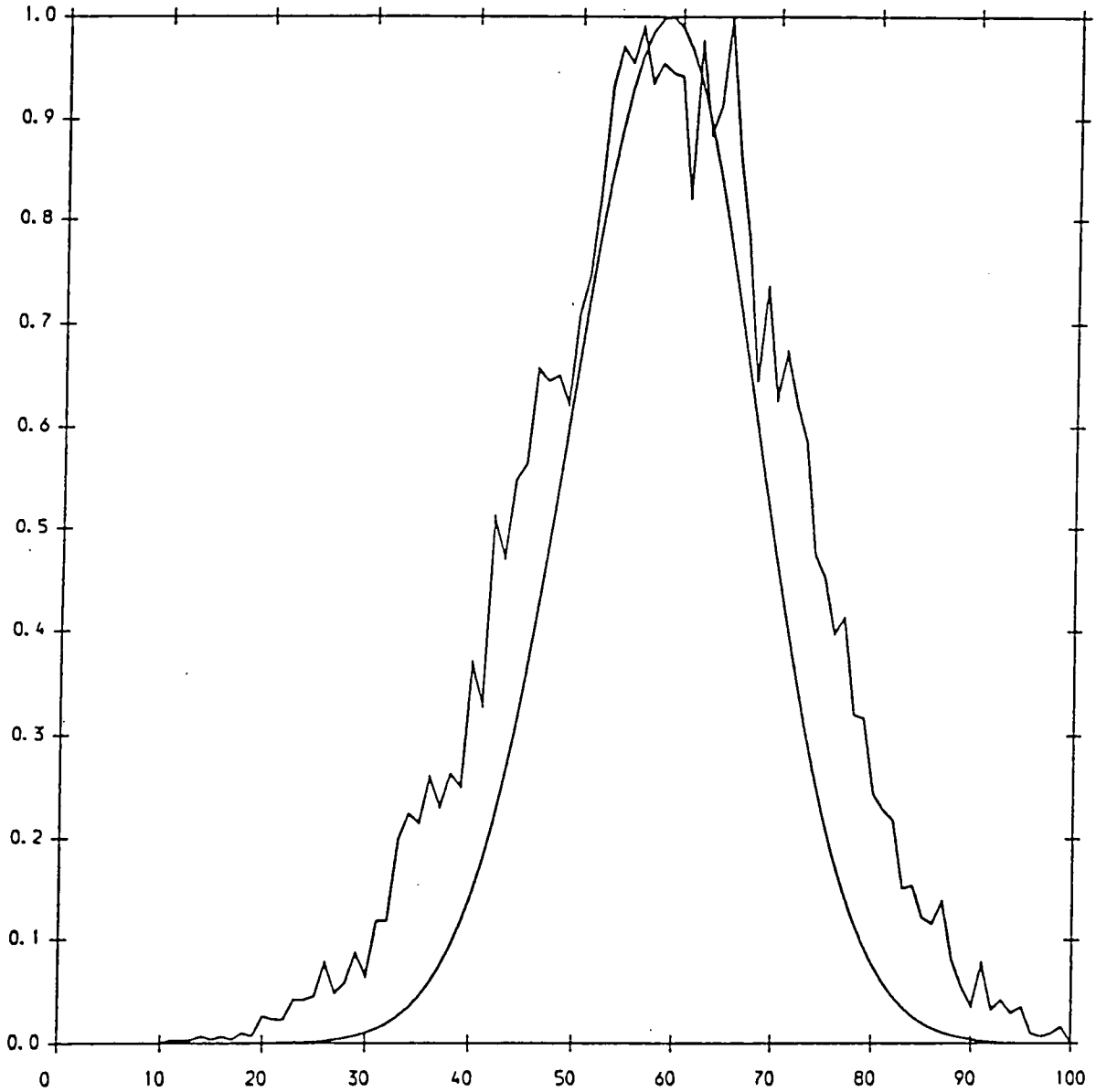


Fig-7.5. The cell rate distribution in the link contains mixed calls with first-order Markov model. The burstiness are 5, 3.33 and 1.67 respectively. Other parameters are the same as in Fig-7.4.

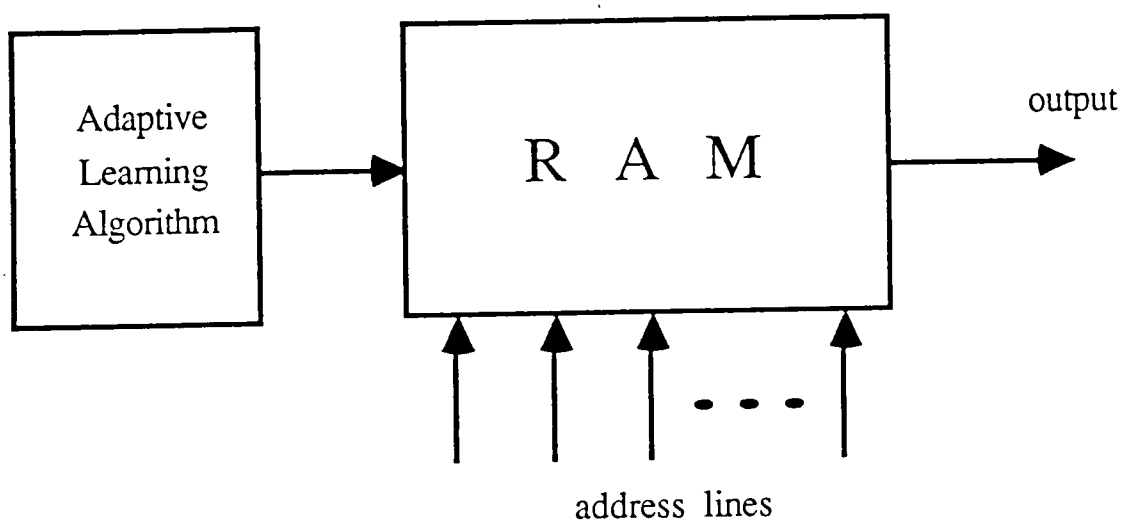


Fig-7.6. RAM implementation of call regulation.

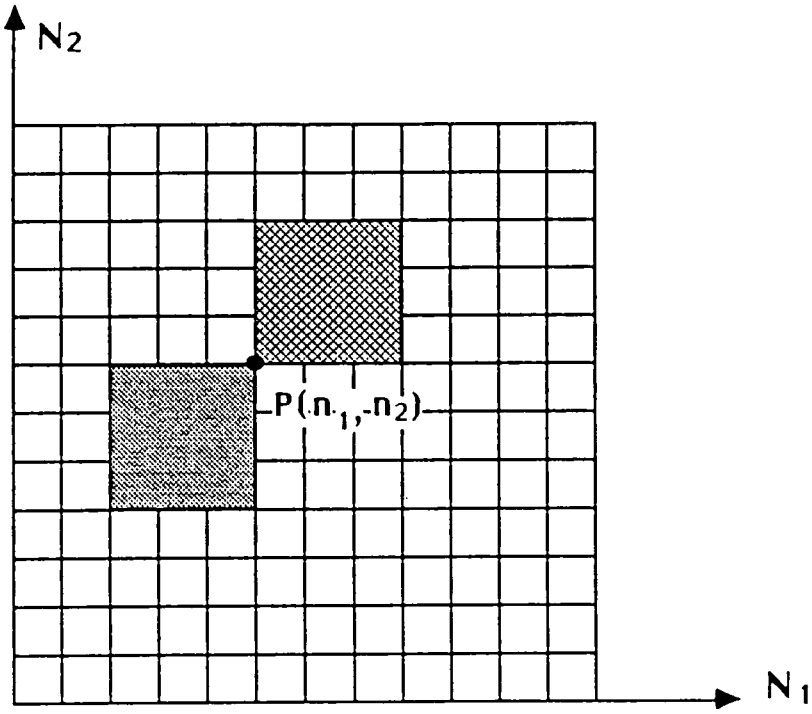


Fig-7.7

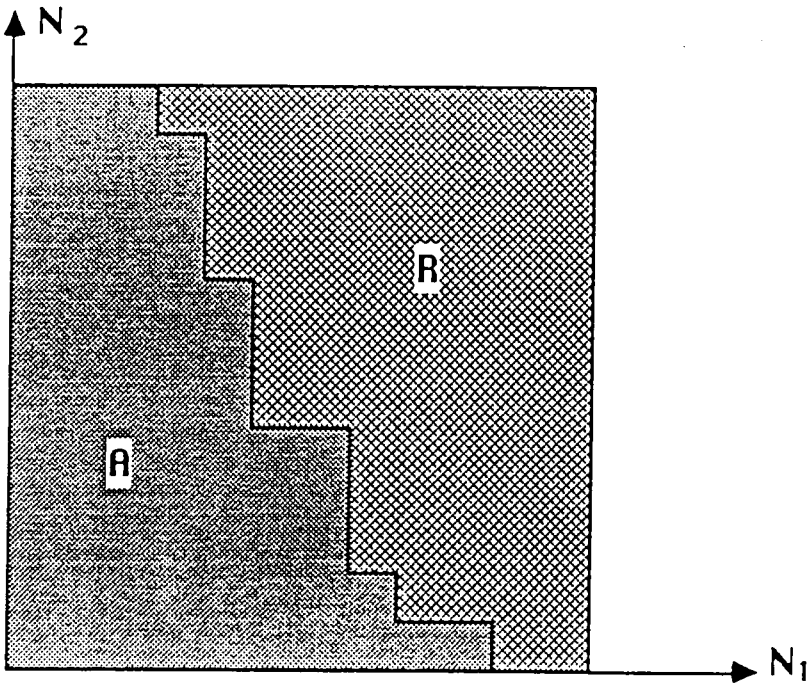


Fig-7.8

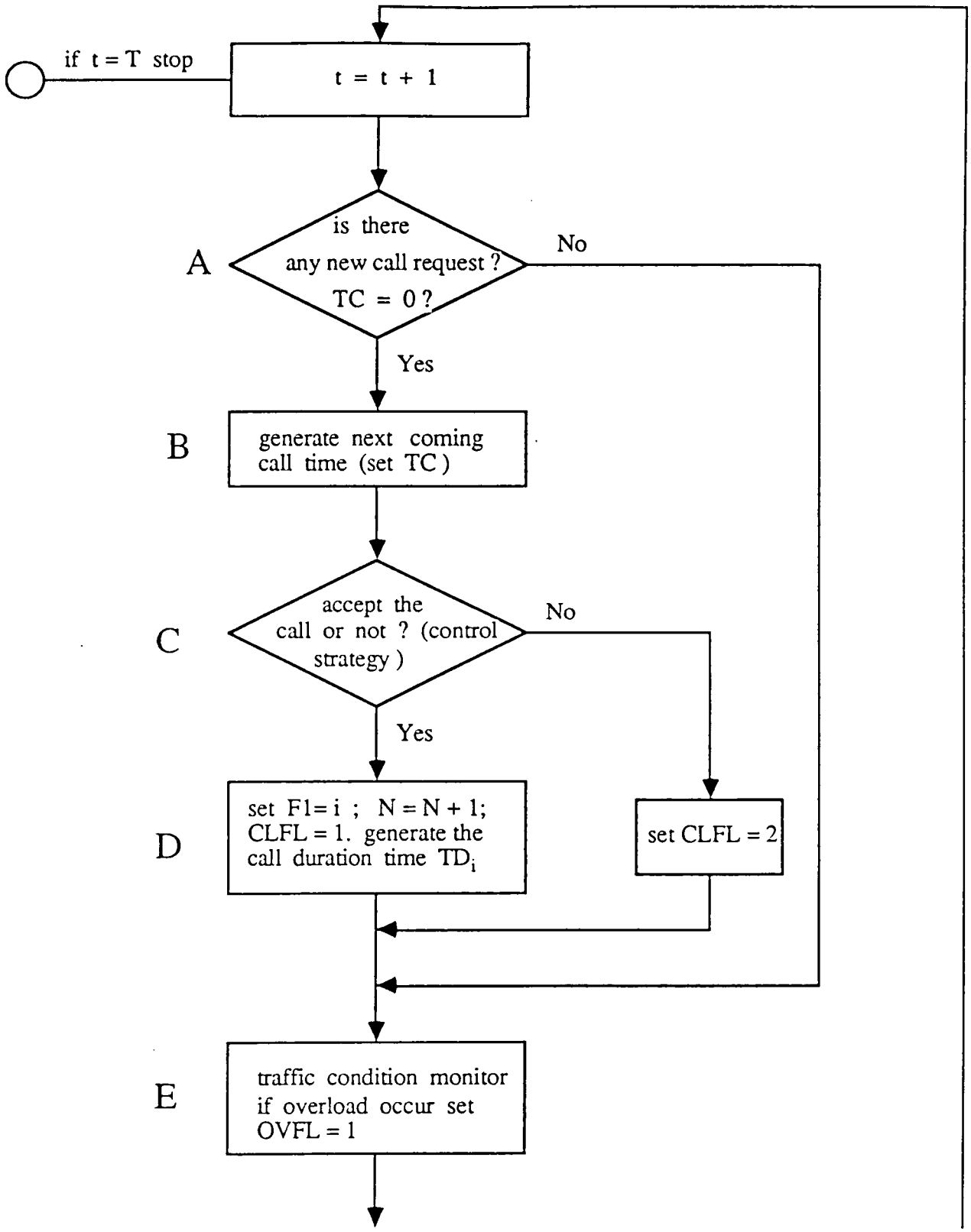


Fig-7.9. The flow chart of the simulation program. Continued on next page.

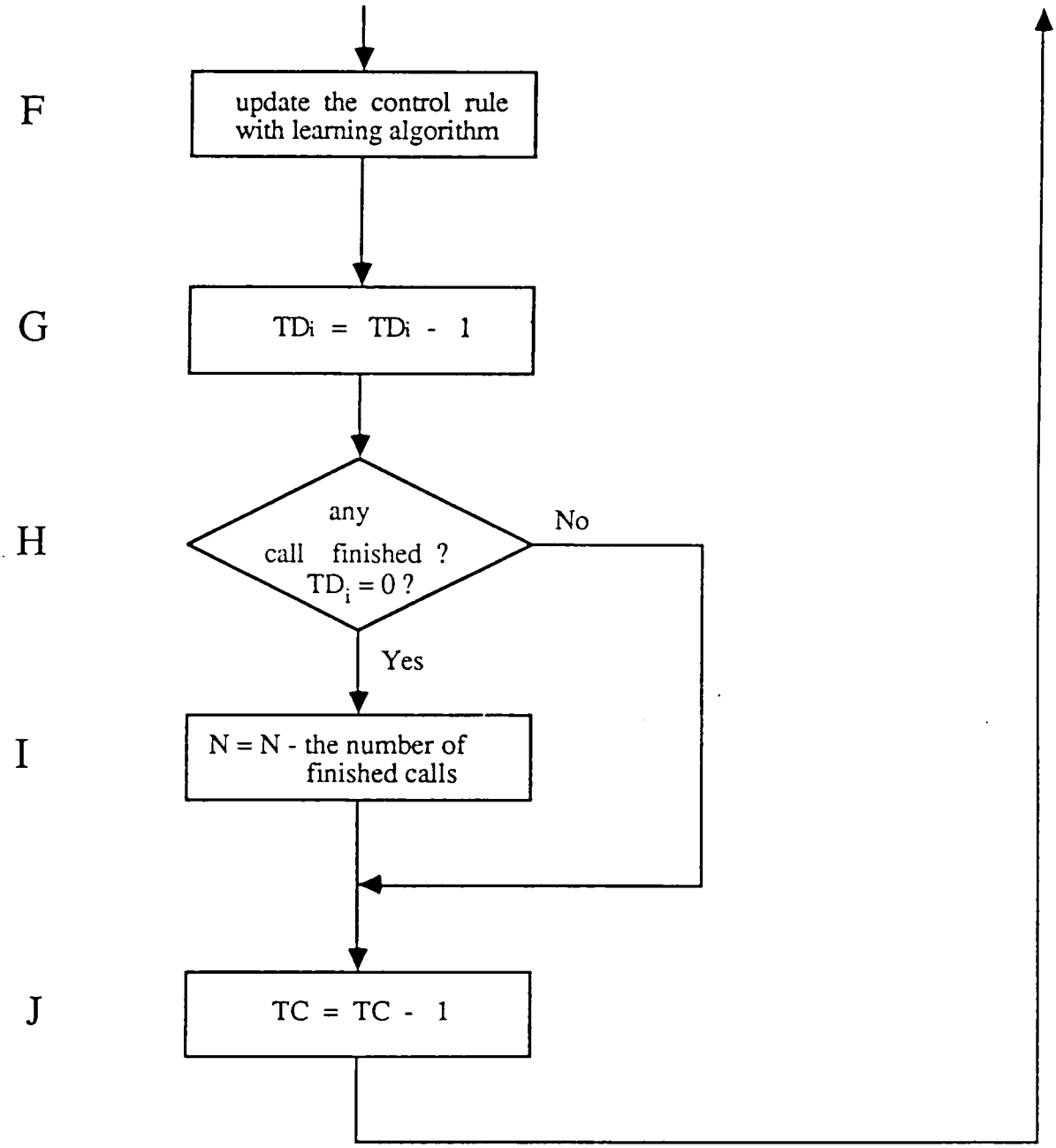


Fig-7.10. The flow chart of the simulation program. Continued from previous page.

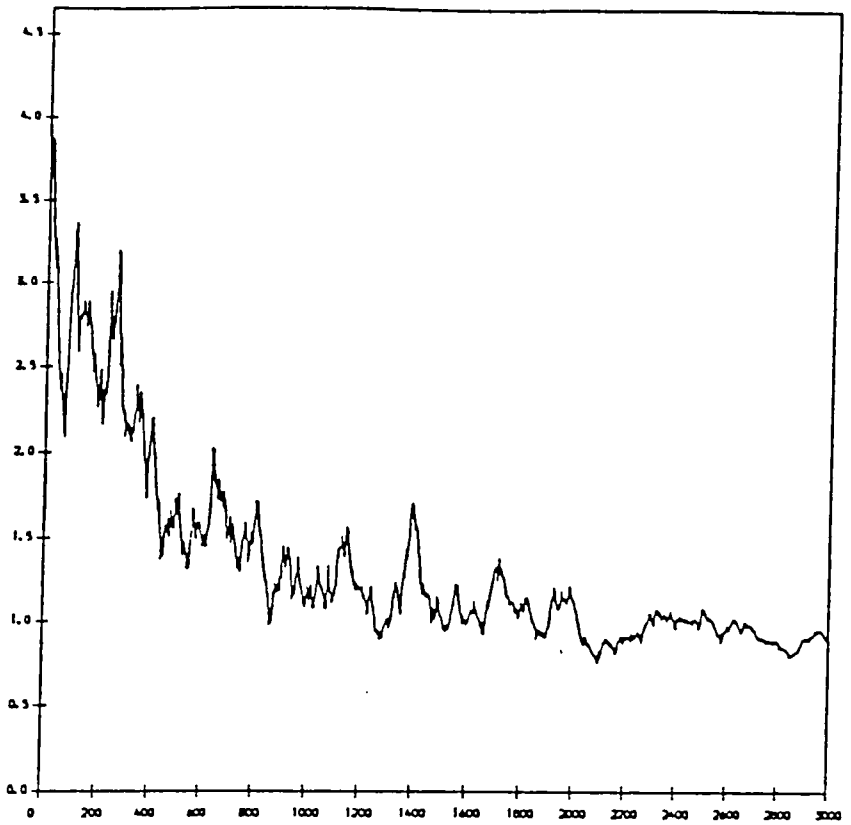


Fig-7.11. Learning curve of $a_1(n)$ with decreasing stepsize.

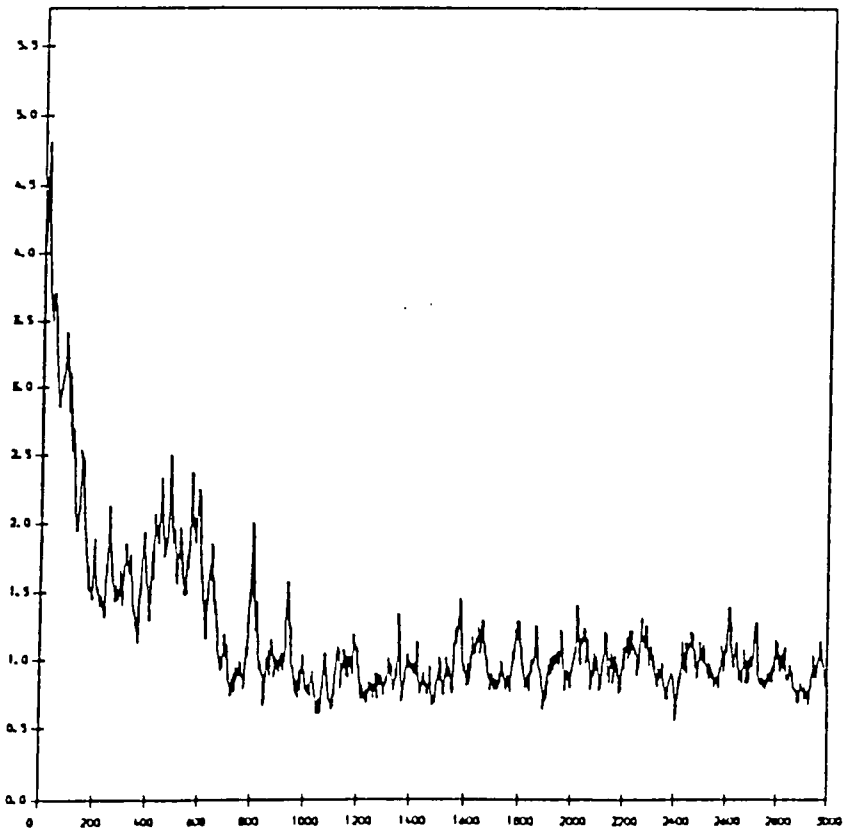


Fig-7.12. Learning curve of $a_1(n)$ with constant stepsize.

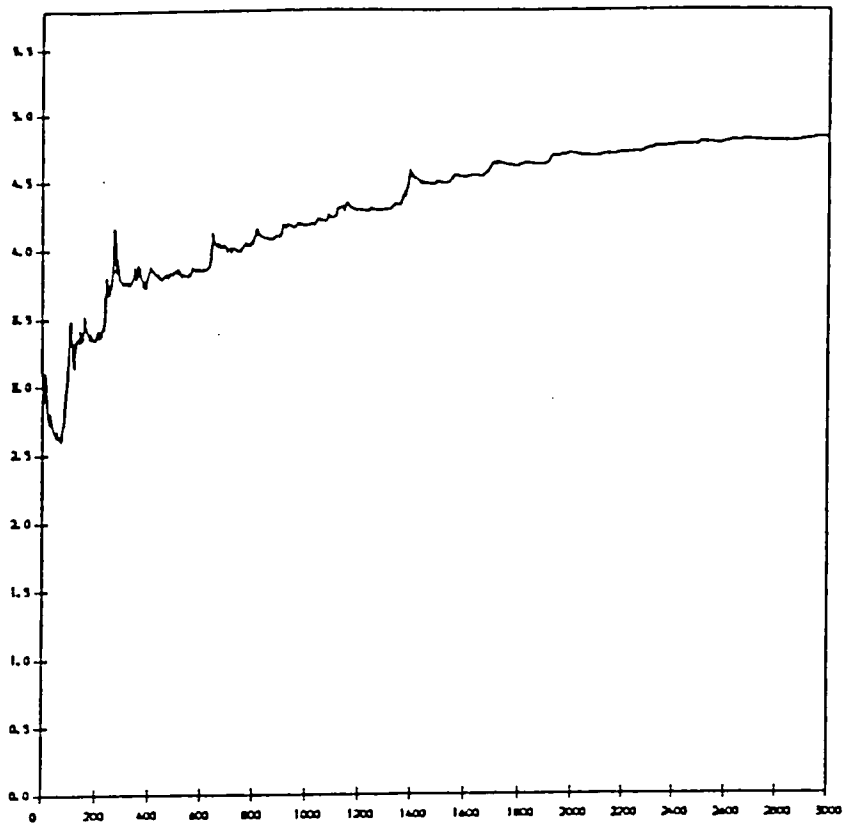


Fig-7.13. Learning curve of $a_2(n)$ with decreasing stepsize.

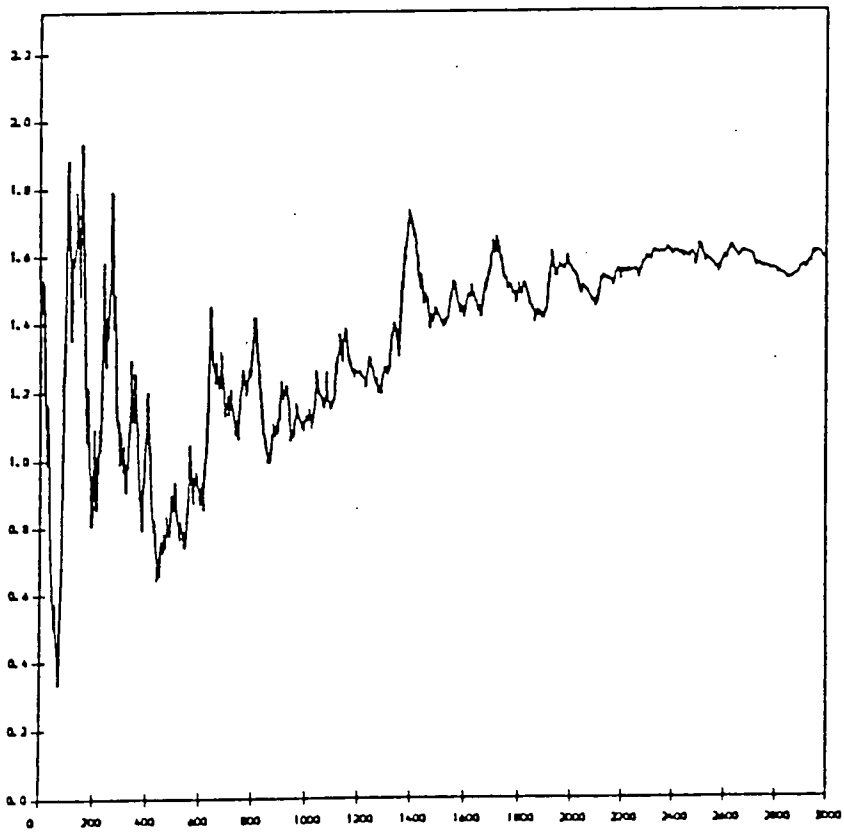


Fig-7.14. Learning curve of $a_3(n)$ with decreasing stepsize.

Chapter Eight
Conclusions and Further Work

From the discussions presented in the previous chapters, this chapter concludes with some overall conclusions and suggestions for further work.

8.1 Convergence Speed of Back-Propagation Algorithm

As the essence of the Back-Propagation algorithm is the hill-climbing or gradient descent algorithm, it is usually slow when the dimensionality is high. Many techniques have been proposed to accelerate its convergence speed. In this thesis two stepsize variation techniques are introduced for speeding up the convergence of Back-Propagation, and the simulation results demonstrated that there is significant improvement. However it is usually very difficult to judge the convergence performance of numerical optimisation algorithms based on pure theoretical analysis, the performance study depends significantly on computer simulation of some bench mark problems. Thus as with numerical optimization algorithms, the convergence performance of various Back-Propagation algorithms are problem specific. The selection of an appropriate algorithm for a specific problem has to be based on empirical knowledge. As the convergence performance of the gradient descent algorithm is closely related to the initial start point value, so it may be fruitful in the future reserach to give more attention to the selection of initial weights. At present in most reported work, the initial weights usually take some small random values. There has been some research using tree classification algorithm to set up initial weights of MLP networks [135]. This can significantly reduce the learning time. A custom-tailored network structure can not only speed up the convergence, but is also benificial for better generalization.

8.2 Generalization

As discussed in Chapter 4 satisfactory generalization usually depends on proper selection of priori constraints and model. In artificial neural networks, the generalization can usually be viewed as an interpolation process. When an input to the network is not included in the learning sample set, the network produces an output by interpolating between known learning samples. The contraints imposed on the network which determines how the network interpolates or generalizes reflect the priori knowledge about the problem. There is a belief that artificial neural networks are able to learn to solve problems which we do not know how to solve. This is obviously excessively optimistic at least at this stage. We have seen in the previous chapters that the generalization performance is closely related to the structure and representation strategy of the network. Thus it is unrealistic to expect artificial neural networks to perform well in situations we know nothing about. It may be more realistic to suggest that in future research more priority should be given to develop function specific modules. These modules can be used as building blocks for large networks. These kind of hierarchical networks may not be as universal as large homogenous networks, like MLP for example, but would be more efficient to train and generalize better.

8.3 Biological Plausibility

One of the main original motivations of research on artificial neural networks was to establish a model of the brain to understand its information processing mechanism. Perhaps the most difficult problem is how to associate neuronal activities in the brain to the high level cognition and thinking process. Artificial neural networks are invaluable in this research. They can be used to verify new hypotheses, which is essential for the development of theory and sometimes difficult to carry out on

biological organisms. So in this sense biological plausibility is very important for neural modeling. However the intricacy of the biochemical processes involved, and the extremely complex connection patterns of biological neural systems means we are still far away from a solid and complete understanding of how the brain works. Many artificial neural networks proposed for neural modeling have some features which can be explored for practical applications. For example they can be used as parallel computing structures to solve engineering problems. It is useful to regard artificial neural networks as a kind of parallel computing structure or a flexible nonlinear adaptive filter which are more efficient than a conventional digital computer on some information processing tasks. From this perspective biological plausibility is not an essential requirement. It can even be sacrificed to suit a specific application if necessary. So far, in most publications about applications of artificial neural networks, they are rarely treated as a strict model of biological neural systems.

8.4 Application Issues

We have discussed in the last paragraph that artificial neural networks can be used as nonlinear adaptive filters. This perspective opens a wide application field for artificial neural networks. It is shown in the previous chapters that artificial neural networks can be used for adaptive equalization, nonlinear system identification and telecommunication call access control. The simulations demonstrated satisfactory results, and was considered to represent a useful extension of adaptive technology.

In Chapter 5 we gave a detailed discussion about the feasibility of using MLP networks for nonlinear system identification. This is a fast developing field in the last few years. Artificial neural networks provide a flexible nonlinear adaptive structure to approximate the underlying nonlinear mechanism of the process to be identified. There are still some theoretical and practical questions which need to be

investigated. One important question is the validation of the input-output difference equation model for nonlinear systems. Validation has been proved under some conditions in the neighbourhood of the equilibrium point. The mathematical neighbourhood could be as large as infinity or as small as indivisible. Thus the validation of the model under practical circumstances still need to be investigated, perhaps by extensive computer simulation and some theoretical analysis. As our knowledge about nonlinear phenomena is very limited, computer simulation is an indispensible tool for nonlinear reserach. However this does not necessarily mean it is not practical to use artificial neural networks for nonlinear system identification. As in most practical identification problems, the true systems are usually never known and identification can only establish an approximation model. Thus it may be more accurate to reformulate the above question as to study how well can an input-output difference equation approximating model a nonlinear system under general conditions.

Besides the MLP networks, radial base function networks have also been proposed for nonlinear system identification [136]. When the centers of the radial base functions are selected before hand, the network coefficients can be calculated by solving a least mean square equation. Thus the learning speed is much faster than MLP networks with a Back-Propagation algorithm. However both MLP and radial base function networks usually require a large sample set to perform well, because good interpolation need a large number of data points. For a small sample set interpolation, some extra structure information is usually needed. In another words, the priori knowledge of the system is needed.

In chapter 6, we discussed the application of ANN for communication channel equalization. This is an example of the application of ANN in digital signal processing. In this case the neural network is used as a nonlinear adaptive struc-

ture. There has also been some research carried out at Durham University on using learning automata for digital filtering [137][138]. However the learning time of the automata increase almost exponentially with increase in the dimensionality of the filter. In this respect, ANN performs much better than learning automata. It may play a significant role in future nonlinear signal processing research.

In summary, we can say that artificial neural networks have a significant potential in applications involving nonlinear adaptive structure.

In this thesis we also discuss the using of ANN for ATM call access control. The advantages of using ANN for control in this case are that it can implement a nonlinear control rule and can learn the control rule from examples. This learning ability or adaptive ability is especially valuable when there is little information about the system operation mechanism available or in a time varying environment. However as discussed in the thesis, ANN usually need a large learning sample set to perform well. This could be a drawback for the application of ANN in control. However ANN can always be used as a last resort in circumstances where the only available information is via input and output observations.

Appendix: Proof of Theorem6.1 and Theorem 6.2

Proof of Theorem6.1:

As $y_i = a_0x_i + a_1x_{i-1} + \cdots + a_nx_{i-n}$, so the point $\mathbf{Y}_i$ in $P_m(1)$ can be represented as

$$\mathbf{Y}_i = \begin{bmatrix} y_i \\ \vdots \\ y_{i-m+1} \end{bmatrix} = \begin{bmatrix} a_0x_i + a_1x_{i-1} + \cdots + a_nx_{i-n} \\ \vdots \\ a_0x_{i-m+1} + a_1x_{i-m} + \cdots + a_nx_{i-n-m+1} \end{bmatrix}$$

and the point $\mathbf{Y}'_i$ in $P_m(-1)$ can be described as

$$\mathbf{Y}'_i = \begin{bmatrix} y'_i \\ \vdots \\ y'_{i-m+1} \end{bmatrix} = \begin{bmatrix} a_0x'_i + a_1x'_{i-1} + \cdots + a_nx'_{i-n} \\ \vdots \\ a_0x'_{i-m+1} + a_1x'_{i-m} + \cdots + a_nx'_{i-n-m+1} \end{bmatrix}$$

The linear separability of $P_m(-1)$ and $P_m(1)$ is equivalent to the existance of a sequence $(b_0,\ b_1,\ \cdots,\ b_{m-1})$ (it can also be denoted as vector $\mathbf{B}$) such that

$$\mathbf{B}^T\mathbf{Y}_i > \mathbf{B}^T\mathbf{Y}'_i$$

that is

$$\mathbf{B}^T(\mathbf{Y}_i - \mathbf{Y}'_i) > 0 \qquad (A1-1)$$

If

$$z_k = x_k - x'_k \quad (k = i,\ i+1,\ \cdots) \qquad C = A \otimes B$$

then $\mathbf{B}^T(\mathbf{Y}_i - \mathbf{Y}'_i)$ can be expressed as

$$\begin{aligned}\mathbf{B}^T(\mathbf{Y}_i - \mathbf{Y}'_i) = {} & b_0(a_0(x_i - x'_i) + a_1(x_{i-1} - x'_{i-1}) + \cdots + a_n(x_{i-n} - x'_{i-n})) \\ & + b_1(a_0(x_{i-1} - x'_{i-1}) + a_1(x_{i-2} - x'_{i-2}) + \cdots + a_n(x_{i-n-1} - x'_{i-n-1})) \\ & + \cdots\cdots \\ & + b_{m-1}(a_0(x_{i-m+1} - x'_{i-m+1}) + a_1(x_{i-m} - x'_{i-m}) + \cdots\end{aligned}$$

$$
\begin{aligned}
&+ a_n(x_{i-n-m+1} - x'_{i-n-m+1}))\\
&= b_0(a_0 z_i + a_1 z_{i-1} + \cdots + a_n z_{i-n})\\
&\quad + b_1(a_0 z_{i-1} + a_1 z_{i-2} + \cdots + a_n z_{i-n-1})\\
&\quad + \cdots\cdots\\
&\quad + b_{m-1}(a_0 z_{i-m+1} + a_1 z_{i-m} + \cdots + a_n z_{i-n-m+1})\\
&= a_0 b_0 z_i + (a_0 b_1 + a_1 b_0) z_{i-1} + \cdots\cdots\\
&\quad + (a_{n-1} b_{m-1} + a_n b_{m-2}) z_{i-n-m+2} + a_n b_{m-1} z_{i-n-m+1}\\
&= c_0 z_i + c_1 z_{i-1} + \cdots + c_{n+m-1} z_{i-n-m+1} \qquad (A1-2)
\end{aligned}
$$

As $x_i = 1$ and $x'_i = -1$, the expression (A1-2) can be written as

$$2c_0 + c_1 z_{i-1} + \cdots + c_{n+m-1} z_{i-n-m+1} \qquad (A1-3)$$

The expression (A1-3) reaches its minimum value when

$$z_k = -2sign(c_k) \quad (k = i-1,\ i-2, \cdots\cdots)$$

The minimum value is

$$2(c_0 - \sum_{k=1}^{n+m-1} |c_k|)$$

So the condition for the inequality (A1-1) to hold is equivalent to

$$2(c_0 - \sum_{k=1}^{n+m-1} |c_k|) > 0$$

that is

$$c_0 > \sum_{k=1}^{n+m-1} |c_k|.$$

We have proved theorem6.1.

Proof of Theorem6.2:

First we assume the A polynomial has a root s which satisfies

$$|s| \geq 1 \quad (|s| \textit{ is the norm of } s)$$

and there exists a B sequence such that

$$c_0 > \sum_{k=1}^{N} |c_k| \quad (N = n + m - 1)$$

As $C = A \otimes B$, the roots of the A polynomial are also the roots of the C polynomial. That is

$$c_0 s^N + c_1 s^{N-1} + \cdots + c_N = 0 \qquad (A1-4)$$

As

$$|c_0 s^N + c_1 s^{N-1} + \cdots + c_N| \geq |c_0 s^N| - |c_1 s^{N-1} + \cdots + c_N|$$

$$|c_1 s^{N-1} + \cdots + c_N| \leq \sum_{k=1}^{N} |c_k s^{N-k}| = \sum_{k=1}^{N} |c_k||s|^{N-k}$$
$$\leq \sum_{k=1}^{N} |c_k||s|^N$$

So

$$|c_0 s^N + c_1 s^{N-1} + \cdots + c_N| \geq |c_0||s|^N - \sum_{k=1}^{N} |c_k||s|^N \qquad (A1-5)$$

From equation (A1-4), we have

$$|c_0 s^N + c_1 s^{N-1} + \cdots + c_N| = 0 \qquad (A1-6)$$

so

$$|c_0||s|^N - \sum_{k=1}^{N} |c_k||s|^N \leq 0 \qquad (A1-7)$$

$$|s|^N(|c_0| - \sum_{k=1}^{N} |c_k|) \leq 0 \qquad (A1-8)$$

as $|s| \geq 1$ and $c_0 > 0$, so

$$c_0 - \sum_{k=1}^{N} |c_k| \leq 0 \qquad (A1-9)$$

that is

$$c_0 \leq \sum_{k=1}^{N} |c_k| \qquad (A1-10)$$

This contradicts the original assumption that

$$c_0 > \sum_{k=1}^{N} |c_k| \qquad (A1-11)$$

Thus the roots of the A polynomial must lie strictly in the unit circle in the complex plane.

If all the roots of A polynomial lie strictly in the unit circle, then the A polynomial can be represented as

$$a_0(z-\alpha_1)(z-\alpha_2)\cdots(z-\alpha_n)$$

and $|\alpha_i| < 1$ for $i = 1,\ 2,\ \cdots,\ n$. Let

$$\alpha = \max_{i=1,2,\cdots,n} |\alpha_i|$$

If we convolve the A sequence with a sequence B_1 which has the characteristic polynomial of

$$sign(a_0)(z+\alpha_1)(z+\alpha_2)\cdots(z+\alpha_n)$$

then we obtain the C_1 polynomial

$$|a_0|(z^2-\alpha_1^2)(z^2-\alpha_2^2)\cdots(z^2-\alpha_n^2)$$

If this process is continued we obtain

$$|a_0|(z^p-\alpha_1^p)(z^p-\alpha_2^p)\cdots(z^p-\alpha_n^p)$$

and

$$\alpha^p < \frac{1}{(n+1)C_n^{[\frac{n}{2}]+1}} \qquad (A1-12)$$

then

$$
\begin{aligned}
c_0 - \sum_{k=1}^{N} |c_k| &= |a_0|(1 - |\alpha_1^p + \alpha_2^p + \cdots + \alpha_n^p| \\
&\qquad - |\alpha_1^p \alpha_2^p + \cdots + \alpha_{n-1}^p \alpha_n^p| \\
&\qquad - \cdots\cdots - |\alpha_1^p \alpha_2^p \cdots \alpha_n^p|) \\
&\geq |a_0|(1 - n\alpha^p - C_n^2 \alpha^{2p} - \cdots - C_n^n \alpha^{np}) \\
&> |a_0|(1 - \frac{1}{n+1} - \frac{1}{n+1} - \cdots - \frac{1}{n+1}) \\
&= |a_0|(1 - \frac{n}{n+1}) > 0 \qquad (A1-13)
\end{aligned}
$$

that is

$$
c_0 - \sum_{k=1}^{N} |c_k| > 0 \quad or \quad c_0 > \sum_{k=1}^{N} |c_k|
$$

The B sequence $(b_0,\ b_1,\ \cdots,\ b_{m-1})$ is the convolution of the $B_1,\ B_2,\ \cdots,\ B_q$ sequences. The coefficients b_i are guaranteed to be real because the coefficients a_i are real. Since the coefficients of the A polynomial are real, any possible complex roots of the polynomial will be in conjugate form. This is also the case for any complex roots of the B polynomial, and all the coefficients b_i will be real. Hence Theorem6.2 is proved.

References

[1] Stevens, C.F. *The Neuron* in *The Brain* A Scientific American Book, W.H.Freeman and Company pp15–25 1979

[2] Feldman, J.A. and Ballard, D.H. *Connectionist Models and Their Properties* Cognitive Science Vol.6 pp205–254 1982

[3] Geschwind, N. *Specializations of the Human Brain* in *The Brain* A Scientific American Book, W.H.Freeman and Company pp108–117 1979

[4] Kalil, R.E. *Synapse Formation in the Developing Brain* Scientific American. December pp38–45 1989

[5] Alkon, D.L. *Memory Storage and Neural Systems* Scientific American. July pp26–34 1989

[6] Lippmann, R *An Introduction to Computing with Neural Nets* IEEE ASSP Magazine April pp4–22 1987

[7] Grossberg, S *Nonlinear Neural Networks : Principles, Mechanisms, and Architectures* Neural Networks Vol.1 pp17–61 1988

[8] Khanna, T *Foundations of Neural Networks* Addison-Wesley Publishing Company 1990

[9] Simpson, P.K. *Artificial Neural Systems : Foundations, Paradigms, Application, and Implementation* Pergamon Press 1990

[10] Widrow, B and Michael, A.L. *30 Years of Adaptive Neural Networks : Per-*

ceptron, Madaline, and Backpropagation Proceedings of the IEEE Vol.78 No.9 September pp1415-1442 1990

[11] Rosenblatt, F *Principles of Neurodynamics* Spartan Books New York 1962

[12] Widrow, B., Mantey, E., Griffiths, L.J. and Goode, B.B. *Adaptive Antenna Systems* Proceedings of IEEE Vol.55 No.12 December pp2143–2159 1967

[13] Widrow, B., Glover, J.R., McCool, J.M., Kaunitz, J., Williams, C.S., Hearn, R.H, Zeidler, J.R., Dong, E. and Goodlin, R.C. *Adaptive Noise Canceling : Principles and Applications* Proceedings of IEEE Vol.63 No.12 December pp1692–1716 1975

[14] Cowan, C.F.N. and Grant, P.M. *Adaptive Filters* Prentice-Hall Signal Processing Series. 1985

[15] Zeidler, J.R. *Performance Analysis of LMS Adaptive Prediction Filters* Proceedings of the IEEE Vol.78 No.12 December pp1781–1806 1990

[16] Minsky, M. and Papert, S.*Perceptrons* MIT Press 1969

[17] Werbos, P *Beyond Regression : New Tools for Prediction and Analysis in the Behavioral Sciences* Ph.D Dissertation Harvard University 1974

[18] Rumelhart, D., Hinton, G. and Williams, R *Learning Representation by Backpropagating Errors* Nature Vol.323 pp533–536 1986

[19] Rumelhart, D. and McClelland, J. *Parallel Distributed Processing : Explorations in the Microstructure of Cognition* Vol.1, Vol.2. MIT Press 1986

[20] Pineda, F *Generalization of Back-Propagation to Recurrent Neural Networks* Physical Review Letters Vol.59 No.19 pp2229–2232 1987

[21] Elman, J.L *Finding Structure in Time* CRL Technical Report 8801 Center for Research in Language, University of California, San Diego April 1988

[22] Williams, R.J. and Zipser, D. *A Learning Algorithm for Continually Running Fully Recurrent Neural Networks* ICS Report 8805. October 1988

[23] Pealmutter, B.A. *Learning State Space Trajectories in Recurrent Neural Networks* CMU-CS-88-191 December 31 1988

[24] Bolz, J., Gilbert, C.D. and Wiesel, T.N. *Pharmacological Analysis of Cortical Circuitry* Trends in Neuroscience Vol.12 No.8 pp292–296 1989

[25] Bellman, R.E. and Dreyfus, S.E. *Applied Dynamic Programming* Princeton University Press 1962

[26] Hinton, G., Sejnowski, T. and Ackley, D. *Boltzmann Machines : constraint satisfaction networks that learn* CMU-CS-84-119 Department of Computer Science Technical Report. Carnegie-Mellon University. 1984

[27] Ackley, D., Hinton, G. and Sejnowski, T. *A Learning Algorithm for Boltzmann Machines* Cognitive Science Vol.9 pp147–169 1985

[28] Aarts, E. and Korst, J *Simulated Annealing and Boltzmann Machines* John Wiley & Sons Ltd 1989

[29] Kirkpatick, S., Gelatt, C.D. and Vecchi, Jr.M.P. *Optimization by Simulated*

Annealing Science Vol.220 No.4598 pp671–680 1983

[30] Stein, D.L. *Spin Glasses* Scientific American July pp36–42 1989

[31] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A. and Teller, E. *Equation of State Calculations by Fast Computing Machines* Journal of Chemical Physics 21 pp1087–1092 1953

[32] Szu, H. and Hartley, R. *Fast Simulated Annealing* Physics Letters. A Vol.122 No.3,4 pp157–162 1987

[33] Sejnowski, T.J. *High-Order Boltzmann Machines* In J.Denker (Ed.), AIP Conference Proceedings 151 : Neural Networks for Computing. New York : American Institute of Physics 1986

[34] Mars, P. *Reforcement Neural Networks and Stochastic Computing* RIPRREP/1000/36/88. 1988

[35] Narendra, K.S. and Thathachar, M.A.L. *Learning Automata* Prentice-Hall 1989

[36] Mars, P. and Narendra, K.S. and Crystall, M.S. *Learning Automata Control of Computer Communication Networks* Proceedings of 3rd Yale Workshop on Adaptive System Theory. pp114–119 1983

[37] Narendra, K.S. and Wheeler, R.M. *Learning Models for Decentralised Decision-Making* Automatica Vol.21 No.4 00479–484 1983

[38] Barto, A.G. *Learning by Statistical Coorperation of Self-Interseted Neuron-Like Computing Elements* Human Neurobiology No.4 pp229–256 1985

[39] Barto, A.G. and Anandan, P *Pattern Recognizing Stochastic Learning Automata* IEEE Transaction on Systems, Man, and Cybernetics. Vol.15 pp360–375 1985

[40] Barto, A.G. and Anandan, P. and Anderson, C.W. *Cooperativity in Networks of Pattern Recognizing Stochastic Learning Automata* Adaptive and Learning Systems. Edited by Kumpati S. Narendra. Plenum Publishing Corporation. pp235–245 1986

[41] Hopfield, J *Neural Networks and Physical Systems with Emergent Collective Computational Abilities* Proceedings of the National Academy of Sciences Vol.79 No.8 pp2554–2558 1982

[42] McElece, R., Posner, E., Rodemich, E. and Venkatesh, S. *The Capacity of the Hopfield Associative Memory* IEEE Transaction on Information Theory Vol.33 No.4 pp461–482 1987

[43] Kohonen, T *Self Organisation and Associate Memory* Third Edition Springer-Verlad 1989.

[44] Gardner, E *The Space of Interactions in Neural Network Models* Journal of Physics A Vol.21 pp257–270 1988

[45] Fontanari, J.F. and Theumann, W.K. *On the Storage of Correlated Patterns in Hopfield's Model* Journal de Physique Vol.51 pp375–386 1990

[46] Kosko, B *Bidirectional Associative Memories* IEEE Transaction on Systems, Man, and Cybernetics. Vol.18 No.1 pp49–60 1988

[47] Kosko, B *Adaptive Bidirectional Associative Memories* Applied Optics. Vol.26, No.23 Dec. pp4947–4960 1987

[48] Haines, K and Hecht-Nielsen, R. *A BAM with Increased Storage Capacity* Proceedings of the IEEE International Conference on Neural Networks Vol-1. pp181-190 1988

[49] Simpson, P *An Algorithm for Encoding an Arbitrary Number of Patterns in a System of Discete Bidirectional Associative Memories* Neural Netwoks Supplement : INNS Abstracts 1 pp220 1988

[50] Rumelhart, D.E. and Zipser, D. *Feature Discovery by Competitive Learning* Cognitive Science 9 pp75–112 1985

[51] Grossberg, S *Competitive Learning : From Interactive Activation to Adaptive Resonance* Cognitive Science 11 pp23–63 1987

[52] Grossberg, S *Adaptive Pattern Classification and Universal Recoding : ii Feedback, Expectation, Olfaction, Illusions* Biological Cybernetics Vol.23 pp187–202 1976

[53] Carpenter, G. and Grossberg, S. *A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine* Computer Vision, Graphics and Image Understanding, 37 pp54–115 1987

[54] Carpenter, G. and Grossberg, S. *Neural Dynamics of Category Learning and Recognition : Attention, Memory Consolidation, and Amnesia* Brain Structure, Learning, and Memory. Davis, J. et al(Ed.) AAAS Symposium Series 1986

[55] Edelman, G.E. and Reeke, Jr.G.N. *Selective Networks Capable of Representative Transformations, Limited Generalizations, and Associative Memory* Proceedings of National Academy of Sciences USA. Vol.79 pp2091–2095 1982

[56] Feldman, J.A., Fanty, M.A., Goddard, N.H. and Lynn, K.J. *Computing with Structured Connectionist Networks* Communications of the ACM Vol.31 No.2 pp170–187 1988

[57] Gallant, S.I. *Connectionist Expert Systems* Communications of the ACM Vol.31 No.2 pp152–169 1988

[58] Shastri, L. *Semantic Networks : An Evidential Formalization and Its Connectionist Realization* Morgan Kaufmann Publishers, Inc. 1988

[59] Marr, D. *Vision* W.H.Freeman and Company 1982

[60] Koch, C. Marroguin, J. and Yuille, A. *Analog 'Neuronal' Networks in Early Vision* Proceedings of the National Academy of Sciences, USA. Vol.83 pp4263–4267 1986

[61] Hopfield, J.J. and Tank, D.W. *Computing with Neural Circuits : A Model* Science Vol.233 August pp625–633 1986

[62] Hopfield, J.J. *Neurons with Graded Response Have Collective Computational Properties Like Those of Two-State Neurons* Proceedings of the National Academy of Sciences, USA Vol.81 pp3088-3092 1984

[63] Tank. D.W. and Hopfield, J.J *Simple 'Neural' Optimization Networks : An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit*

IEEE Transactions on Circuits and Systems, Vol.33 No.5 pp533–541 1986

[64] Lee, B.W. and Sheu, B.J. *Modified Hopfield Neural Networks for Retrieving the Optimal Solution* IEEE Transactions on Neural Networks Vol.2 No.1 pp137–141 1991

[65] Mead, C. *Analog VLSI and Neural Systems* Addison-Wesley Publishing Company 1989

[66] Lyon, R.F. and Mead, C. *An Analog Electronic Cochlea* IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol.36 No.7 pp1119–1134 1988

[67] Mahowald, M.A. and Mead, C. *A Silicon Model of Early Visual Processing* Neural Networks, Vol.1 pp91–97 1988

[68] Hutchinson, J., Koch, C., Luo, J. and Mead, C. *Computing Motion Using Analog and Binary Resistive Networks* COMPUTER March pp52–63 1988

[69] Rosenblatt. F *Principles of Neurodynamics* Spartan Books, New York 1962

[70] Bedworth, M. and Bridle, J.S. *Experiments with the Back-Propagation Algorithm : A systematic look at a small problem* RIPRREP /1000/9/87

[71] Webb, A.R., Lowe, D. and Bedworth, M.D. *A Comparison of Nonlinear Optimization Strategies for Feed-Forward Adaptive Layered Network* Royal Signals and Radar Establishment Memorandum 4157 1988

[72] Becker, S. and Cun, Y.L. *Improving the Convergence of Back-Propagation Learning with Second Order Methods* Proc 1988 Summer School on Connec-

tionist Model, Carnege-Mellon University. pp29-37

[73] Fahlman, S.E. *Fast-Learning Variation on Back-Propagation : An Empirical Study* Proc 1988 Summer School on Connectionist Model. Carnege-Mellon University. pp38–51

[74] Battiti, R *Optimization Methods for Back-Propagation : Automatic Parameter Tuning and Faster Convergence* Proceedings of IJCNN-90-WASH-DC January 1990. Lawrence Erlbaum Associates, Publishers. pp593–596 Vol.1

[75] Vogl, T.P., Mangis, J.K., Zink, W.T. and Alkon, D.L. *Accelerating the Convergence of the Back-Propagation Method* Biological Cybernetics Vol-59 pp257–263 1988

[76] Haykin, S *Adaptive Filter Theory* Englewood Cliffs NJ, Prentic-Hall 1986

[77] Jacobs, R.A. *Increased Rates of Convergence Through Learning Rate Adaptation* Neural Networks Vol-1 pp195–207 1988

[78] Lloyd, P.J. *Guidlines for Implementing Problems on a Multi-Layer Perceptron Network* RIPRREP /1000/27/88

[79] Tesauro, G. and Janssens, B. *Scaling Relationships in Back Propagation Learning* Complex System vol-2 No.1 pp38–44 1988

[80] Sejnowski, T.J. and Rosenberg, C.R. *Parallel Network that Learn to Pronounce English Text* Complex System vol-1 pp145–168 1987

[81] Powell, M.J.D. *Convergence Properties of Algorithms for Nonlinear Optimiza-*

tion SIAM Review vol-28 No.4 pp487–500 1986

[82] Proceedings of IJCNN-90-WASH-DC January 1990. Lawrence Erlbaum Associates, Publishers.

[83] Holland, J.H., Holyoak, K.J., Nisbett, R.E. and Thagard, P.R. *Induction : Processes of Inference, Learning, and Discovery* The MIT Press 1987

[84] Bloom, F.E., Lazerson, A. and Hofstadter, L. *Brain, Mind, and Behavior* W.H.Freeman and Company 1985

[85] Broomhead, D.S. and Lowe, D. *Multivariable Functional Interpolation and Adaptive Networks* Complex Systems vol-2 pp321–355 1988

[86] Wolpert, D.H. *Contructing a Generalizer Superior to Nettalk via a Mathematical Theory of Generalization* Neural Networks Vol-3 No.4 pp445–452 1990

[87] Baum, E.B. and Haussler, D. *What Size Net Gives Valied Generalization ?* Neural Computation Vol-1 pp151-160 1989

[88] Denker, J., Schwartz, D., Wittner, B., Solla, S., Howard, R., Jackel, L. and Hopfield, J. *Large Automatic Learning, Rule Extraction, and Generalization* Complex Systems Vol-1 pp877–922 1987

[89] Funahashi, K.I. *On the Approximate Realization of Continuous Mapping by Neural Networks* Neural Networks Vol-2 No.3 pp183–192 1989

[90] Barto, A.G. *Connectionist Learning for Control : An Overview* preprint

[91] Judd, S. *Learning in Networks is Hard* Proceedings of the IEEE First Interna-

tional Conference on Neural Network 1986. pp685–700

[92] Nadel, L., Cooper, L.A., Culicover, P. and Harnish, R.M. editors *Neural Connections, Mental Computation* A Bradforf Book, The MIT Press 1989

[93] Feldman, J.A. *Structured Neural Networks in Nature and in Computer Science* in *Neural Computers* edited by Rolf Eckmiller and Christoph v.d. Malsburg pp17–21 Springer-Verlag 1988

[94] Desimone, D. and Ungerleider, L.G. *Multiple Visual Area in Caudal Superior Temporal Sulcus of the Macaque* The Journal of Comparative Neuralogy 248 pp164–189 1986

[95] Cun, Y.le. *Generalization and Network Design Strategies* Technical Report CRG-TR-89-4 Dept. of Computer Science, University of Toronto. June 1989

[96] Duda, R.D and Hart, P.E *Pattern Classification and Scene Analysis* John Wiley & Sons, Inc. 1973

[97] Saund, E *Dimensionality-Reduction Using Connectionist Networks* IEEE Transaction on Pattern Analysis and Machine Intelligence, vol-11, No.3, pp304–314 1989

[98] Bourard, H and Kamp, Y *Auto-association by Multilayer Perceptrons and Singular Value Decomposition* Biological Cybernetics, vol-59, pp291–294 1988

[99] Baldi, P and Hornik, K *Neural Networks and Principal Component Analysis : Learning from Examples Without Local Minima* Neural Networks, vol-2, pp53–58 1989

[100] Arozullah, M and Namphol, A *Neural Network Based Data Compression Using Scene Quantization* Proceedings of International Joint Conference on Neural Networks, Jan, Vol-2, pp241–244 1990

[101] Elman, JL, Zipser, D *Learning the Hidden Structure of Speech* Journal of Acoust.Soc.Am. vol-83, pp1615–1626 1987

[102] Cover, T.M. *Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition* IEEE Transactions on Electronic Computers. June pp326–334 1965

[103] Norton, J.P. *An Introduction to Identification* Academic Press Inc. 1986.

[104] Ljung, L *System Identification : Theory for the User* Prentice-Hall Information and System Sciences Series 1987

[105] Schetzen, M *The Volterra and Wiener Theories of Nonlinear Systems* John Wiley & Sons, Inc. 1980

[106] Gallman, P.G *An Iterative Method for the Identification of Nonlinear Systems Using the Uryson Model* IEEE Trans. AC-20 pp567–578 1975

[107] Leontaritis, I.J. and Billings, S.A. *Input-output parametric models for non-linear systems Part I : deterministic non-linear systems* International Journal of Control. Vol-41 No.2 pp303–328 1985

[108] Narendra, K.S. and Parthasarathy, K *Neural Networks and Dynamical Systems* Part II : Identification. Yale University Technical Report. Report No. 8902 February 1989.

[109] Anderson, B.D.O. *Exponential stability of linear equations arising in adaptive identification* IEEE Transcation on Automatic Control, AC-22 1977 p83-88

[110] Bai, E.W. and Sastry, S *Persistency of excitation, sufficient richness and parameter convergence in discrete time adaptive control* Systems Control Letter., 6, 1985 p153-163

[111] Kailath, T *Linear Systems* Prentice-Hall, Inc. 1980

[112] Thompson, J.M.T. and Stewart, H.B. *Nonlinear Dynamics and Chaos* John Wiley & Sons, Ltd. 1986

[113] Ruelle, D. *Strange Attractors* The Mathematical Intelligencer 2, 1980 p126-137

[114] Farmer, J.D and Sidorowich, J.J *Predicting Chaotic Time Series* Physical Review Letters Vol-59 No.8 1987 pp845–848

[115] Roden, M.S *Digital and Data Communication Systems* Prentice–Hall Inc. 1982

[116] Qureshi, S.U.H. *Adaptive Equalization* Proceeding of the IEEE, 73, 9, pp1349–1387 1985

[117] Johnson, C.R.Jr. *Admissibility in Blind Adaptive Channel Equalization* IEEE Control Systems Magazine Vol-11 No.1 pp3–15 1991

[118] Gibson,G.J, Siu, S and Cowan, C.F.N *Multilayer Perceptron Structures Applied to Adaptive Equalizers for Data Communication* Proceedings of ICASSP-89, IEEE Press, pp1183–1186 1989

[119] Jha, S.K, Soraghan, J.J and Durrani, T.S *Equalization Using Neural Networks*

Proceedings of First IEE International Conference on Artificial Neural Networks, pp356-360 1989

[120] Siu, S, Gibson, G.J and Cowan, C.F.N *Decision Feedback Equalization Using Neural network Structures* Proceedings of First IEE International Conference on Artificial Neural Networks, pp125–128 1989

[121] Kohonen,T *Self-organization and Associative Memory* Third Edition Springer–Verlag 1989

[122] Minzer, S.E *Broadband ISDN and Asynchronous Transfer Mode (ATM)* IEEE Communication Magazine September 1989 pp17–24

[123] *Broadband aspects of ISDN* CCITT Study Group XVII Draft Recommendation I.121, Temporary Doc. 49. Feb.1988

[124] *Broadband aspects of ISDN-Baseline document* T1S1 Tech. Subcommittee Doc. no. T1S1.1/89-395. Sept. 1989.

[125] Woodruff, G.M and Jositpaiboon, R *Multimedia Traffic Management Principles for Guaranteed ATM Network Performance* IEEE Journal on Selected Areas in Communications. Vol.8. No.3. April 1990. pp437–446

[126] Verbiest, W, Pinnoo, L and Voeten, B *The Impact of the ATM Concept on Video Coding* IEEE Journal on Selected Areas in Communications. Vol.6. No.9. December 1988. pp1623–1632

[127] Ohnishi H, Okada T and Noguchi K *Flow Control Schemes and Delay/Loss Tradeoff in ATM Networks* IEEE Journal on Selected Areas in Communication.

Vol.6 No.9 Dec. 1988. pp1609–1616

[128] Gallassi G, Rigolio G and Verri L *Rescouce Management and Dimensioning in ATM Networks* IEEE Network Magazine May 1990. pp8–17

[129] Narendra K.S and Mars P *The Use of Learning Algorithms in Telephone Traffic Routing–A Methodology* Automatica Vol.19 1983 pp495–502

[130] Hiramatsu A *ATM Communication Network Control by Neural Network* IEEE Transaction on Neural Networks Vol.1 No.1 1990 pp122–130

[131] Parzen, E *Modern Probability Theory and Its Applications* John Wiley & Sons, Inc 1960

[132] Griffiths, T.R *Analysis of Connection Acceptance in Asynchronous Transfer Mode Networks* Procceedings of 7th IEE teletraffic Symposium Durham 1990

[133] Hui, J.Y *Resource Allocation for Broadband Networks* IEEE Journal on Selected Areas in Communications. Vol.6 No.9 Dec. 1988. pp1598–1608

[134] Widrow. B and Stearns. S.D *Adaptive Signal Processing* Englewood Cliffs, NJ: Prentice-Hall. 1985

[135] Sethi. I.K. *Entropy Nets : From Decision Trees to Neural Networks* Proceedings of IEEE Vol.78 No.10 1990 pp1605–1613

[136] Poggio. T and Girosi. F *Networks for Approximation and Learning* Proceedings of IEEE Vol.78 No.9 1990 pp1481–1497

[137] Tang, C.K.K and Mars, P. *Stochastic Learning Automata and Adaptive IIR*

Filters IEE Proc. F. Vol-138 No.4 Aug. 1991 pp331-340

[138] Nambiar, R *Reinforcement Learning Algorithms and Adaptive Digital Filters* Technical Report of School of Engineering and Applied Science at University of Durham. Dec. 1990

Publications

Appeared on IEE Eighth U.K. Teletraffic Symposium 10-12 April 1991.

Adaptive ATM Call Access Control Using Learning Algorithms

J.R.Chen and P.Mars †

1. Introduction

The Broadband Integrated Services Digital Network (B-ISDN) is an emerging communication network which is intended to provide multimedia services to its customers in a flexible and cost-effective manner. The services include voice, video and data transmission. Research and development in B-ISDN is a very active area.

The traditional transport paradigm used for ISDN is synchronous transfer mode (STM) [1]. The rule for subdivision and allocation of bandwidth using STM is to allocate time slots within a recurring structure (frame) to a service for the duration of call. An STM channel is identified by the position of its time slots within a synchronous structure. The hierarchical channel structure of STM consists of several bearer channels, and each of them has different transmission rate. One of the drawbacks of appling STM to B-ISDN is its rigid bearer channel structure which make the dynamic allocation of time slots difficult [1]. In an B-ISDN enviroment, the services have greatly varied bit rate, and some kind of dynamic allocation of time slots (or bandwidth) is necessary to make efficient use of the bandwidth resource. Thus the asynchronous transfer mode (ATM) has attracted significant attention as a transport paradigm for B-ISDN [2] [3]. In ATM, specific periodic time slots are not assigned to a fixed service, usable bandwidth is segmented into fixed size information bearing units called packets or cells. Each cell consists of a header and an information field. These cells can be dynamically allocated to services on demand. In comparison to STM, ATM is more flexible, and may have potential gain in bandwidth efficiency by buffering and statistically multiplexing bursty traffic at the expense of cell delay and loss [4]. In this paper we concentrate on the statistical multiplexing control strategy and consider two access control strategies based on learning algorithms. We first discuss the basic problem of bandwidth resource manegement in ATM. Two new adaptive strategies are then considered with associated simulation results and a critical discussion.

2. Bandwidth Resource Management of ATM

To guarantee performance requirements like cell delay and loss demanded by the services which are supported by the B-ISDN, a bandwidth allocation strategy (or traffic control strategy, call regulation) must be implemented in the transport layer to control the multiplexing of cells. Generally, there are two call regulation rules [4]. One is nonstatistical multiplexing, in which if the sum of the peak cell rate of all the hold on calls (include the new incoming call) does not exceed the output link rate, then the new incoming call is accepted. Otherwise it would be rejected. That is the call accept condition is

$$\sum_i P_i \leq C$$

† The authors are with the School of Engineering and Applied Science University of Durham UK

where P_i is the peak rate of the ith hold on call, and C is the capacity of the output link at the node. This approach is quite similar to bandwidth reservation for STM, but with the added flexibility of being able to reserve any peak rate required, rather than a multiple of a base channel rate. The strong advantage with nonstatistical multiplexing is minimal cell delay and no cell loss due to buffer overflow. However in the case when a large proportion of the traffic flow in the link is bursty, the nonstatistical multiplexing can show low efficiency in making use of bandwidth resource. Thus statistical multiplexing is considered to exploit the burstiness of traffic flow and obtain potential gain in bandwidth efficiency. In statistical multiplexing, the total peak cell transmission rate of all the accepted calls is allowed to exceed the capacity of the link at the expense of cell delay or cell loss. However under a proper control strategy the cell delay or cell loss can be controlled within a tolerable range.

Statistical multiplexing can only increase bandwidth efficiency under certain conditions. The preconditions are that the average burst length B of calls is short, the peak rate to link capacity ratio (PLR) of calls is low and the burstiness of calls are high [4]. Let P denote the peak rate of a call, A the average rate and C the capacity of the link, then the burstiness of the call is defined as P/A, and $PLR = P/C$. In [4] the authors give some computer simulation results on the feasibility of using statistical multiplexing in homogeneous traffic and heterogeneous traffic enviroments. In the homogeneous traffic case, generally PLR should be less than 0.1. These preconditions can be met in many cases in B-ISDN due to wide bandwidth and inherently bursty data services. Also advanced image coding technique are making the traditional continuous sources like video into bursty sources [5].

3. Adaptive Statistical Multiplexing Strategies

Adaptive call regulation can use learning automata [6] or an artificial neural network [7] as its basic structure. The neural network controller considered in [7] used the back-propagation algorithm for training. Thus it has a potential local minima problem and retraining is not easy. Another undesirable feature of this controller is that it only uses the incoming cell pattern in the link as the basis for call regulation and does not take into account the fact that different type calls may require different bandwidth. Sometimes the traffic condition on a link may not be able to support a wideband service but is adequate for a narrowband call. This kind of situation cannot be dealt with efficiently by the neural network controller mentioned above. If the traffic characteristics can be approximated by a normal distribution, a simple linear call regulation rule may be used. In the following we discuss a perceptron like adaptive call regulation controller which uses a linear inequality as decision rule.

(a) Perceptron Control Rule

Consider an adaptive call regulation rule based on the inequality

$$a_1 N_1 + a_2 N_2 + a_3 N_3 < T$$

where N_1 is the number of hold on calls of narrowband calls, and N_2 and N_3 are for intermediate-band and wideband calls. Each class of call has different cell rates. Of course one can classify calls into more classes, but for the simplicity of control rule, large number of classes are unfavorable. The coefficients a_1, a_2 and a_3

can be adaptively updated using a perceptron like algorithm [8] as follows.

If a call request is accepted but the following cell loss rate exceeds the performance requirement, then

$$a_1(n+1) = a_1(n) + \alpha N_1 \quad a_2(n+1) = a_2(n) + \alpha N_2 \quad a_3(n+1) = a_3(n) + \alpha N_3$$

where α is the learning stepsize. If the call is rejected and the cell loss rate is much lower than the performance requirement, then

$$a_1(n+1) = a_1(n) - \beta N_1 \quad a_2(n+1) = a_2(n) - \beta N_2 \quad a_3(n+1) = a_3(n) - \beta N_3$$

where β is the learning stepsize.

(b) RAM Map Control Rule

Both the perceptron mentioned above and the neural network controller considered in [7] essentially implement a functional mapping. If we consider that the output of the controller is just accept or reject and the input can usually be transformed into binary form, these mappings are just Boolean logic functions. and can be implemented using digital logic circuits If variables N_1, N_2 and N_3 are viewed as three orthogonal axes, then every combination of hold on call pattern in a link is represented as an integer point in the space. All these points form a lattice array with finite nodes. This lattice array can be implemented by a RAM. N_1, N_2 and N_3 are represented as binary numbers to the address lines, the output is single bit with 1 representing accept and 0 for reject. For a RAM with 16 address lines, it contains a $2^{16} = 65536$ node lattice, and is sophisticated enough for most applications. To train this RAM network, a learning paradigm which is similar to the self-organization map algorithm [9] is introduced.

4. Simulation Results and Discussion

For the perceptron control rule discussed, the $\alpha - LMS$ learning algorithm is used for training. To verify the analytical prediction, the homogeneous traffic situation was simulated, and the simulation results are shown below. They are obtained with a perceptron control rule.

N (Average number of on hold calls)	p_{ov}	Efficiency	Theoretical Estimation
154	0.000389	73.9%	78.8%
75	0.000552	70.3%	72.8%
47	0.001952	69.9%	70.4%

Table-1

The simulation results for a heterogeneous traffic situation are shown in Table-2 and Table-3. The composition of incoming call flow is 60% narrowband calls, 30% mediumband calls and 10% wideband calls. Table-2 shows the simulation results for short burst call source which has the average burst length 4.25 cells and the burstiness of 2. Table-3 shows the simulation results for long burst call source which has the average burst length 9.5 cell and burstiness of 2. There are two ways to measure the bandwith efficiency. One is to use the utilization of the capacity of the link like that defined in (4), and the other way is to measure the

average peak cell rate or the actual throughput in the link. Both of these parameters are listed in the tables and the capacity of the link is normalized to 100.

Control strategy	Efficiency	Average peak rate	p_{ov}	Cell loss rate
nonstatistical mutiplexing	41.4%	97	0	0
perceptron control rule	54.8%	115	0.0018	7.57×10^{-5}
perceptron control rule	59.8%	121	0.0005	1.75×10^{-5}
RAM map control rule	57.7%	116	0.0016	5.25×10^{-5}
RAM map control rule	62.1%	123	0.0050	1.68×10^{-4}

Table-2

Control strategy	Efficiency	Average peak rate	p_{ov}	Cell loss rate
nonstatistical multiplexing	44.3%	97	0	0
perceptron control rule	61.7%	123	0.0008	3.85×10^{-5}
perceptron control rule	56.8%	122	0.00006	1.86×10^{-6}
RAM map control rule	67.6%	134	0.0062	2.12×10^{-4}
RAM map control rule	68.8%	138	0.0053	1.79×10^{-4}

Table-3

5. Conclusion

In comparison to STM, ATM has more fexibility in bandwidth resource mangement. To explore the potential gain in bandwidth usage efficiency which is made possible by ATM, statistical mltiplexing is needed in call access control. Of course there are some preconditions which are necessary for profitable use of statistical multiplexing. The calls must have high burstiness and low PLR (peak cell rate to link capacity ratio). Under these conditions, statistical multiplexing with the learning algorithms discussed in this report can provide an obvious gain in bandwidth efficiency. This has been demonstrated by simulation results. Further work is needed on more intensive simulation studies and eventually testing on real ATM communication networks.

References:

[1] Minzer, S.E *Broadband ISDN and Asynchronous Transfer Mode (ATM)* IEEE Communication Magazine September 1989 pp17–24

[2] *Broadband aspects of ISDN* CCITT Study Group XVII Draft Recommendation I.121, Temporary Doc. 49. Feb.1988

[3] *Broadband aspects of ISDN-Baseline document* T1S1 Tech. Subcommittee Doc. no. T1S1.1/89-395. Sept. 1989.

[4] Woodruff, G.M and Jositpaiboon, R *Multimedia Traffic Management Principles for Guaranteed ATM*

Network Performance IEEE Journal on Selected Areas in Communications. Vol.8. No.3. April 1990. pp437–446

[5] Verbiest, W, Pinnoo, L and Voeten, B *The Impact of the ATM Concept on Video Coding* IEEE Journal on Selected Areas in Communications. Vol.6. No.9. December 1988. pp1623–1632

[6] Narendra K.S and Mars P *The Use of Learning Algorithms in Telephone Traffic Routing–A Methodology* Automatica Vol.19 1983 pp495–502

[7] Hiramatsu A *ATM Communication Network Control by Neural Network* IEEE Transaction on Neural Networks Vol.1 No.1 1990 pp122–130

[8] Minsky, M *Perceptrons* Expanded Edition The MIT Press 1988

[9] Kohonen, T *The Self-Organization Map* Proceedings of the IEEE Vol.78 No.9 September 1990 pp1464–1480

Appeared on Proceedings of IEEE Colloquium on Neural Networks in Control and Modelling of Industrial Processes. Polytechnic of Central London, April 1991.

Some Aspects of Non-Linear System Identification Using Neural Networks

J.R.Chen and P.Mars

School of Engineering and Applied Science

University of Durham UK

System identification, and in particular non-linear system identification, is an important problem for systems control. In this paper we discuss some fundamental constraints in using MLP neural networks for non-linear system identification.

Recently there has been considerable interest reported in the use of neural networks for system control and identification [1] [2] [3]. A significant number of nonlinear dynamic systems can be described by the recursive equation

$$x_{n+1} = f(x_n, x_{n-1}, \ldots, x_{n-p+1}, u_n) \tag{1}$$

where x_i is the output of the system and u_n is the input excitation. In this paper we discuss the identification problem for two classes of nonlinear systems described by

$$x_{n+1} = a_0 x_n + a_1 x_{n-1} + \ldots + a_p x_{n-p+1} + f(u_n) \tag{2}$$

and

$$x_{n+1} = f(x_n, x_{n-1}, \ldots, x_{n-p+1}) + u_n \tag{3}$$

Equation (2) and (3) are called Model-I and model-II nonlinear systems respectively[1]. The essence of applying a neural network to nonlinear system identification is to use the neural network to approximate the nonlinear mapping f( ) in (1). Theoretically, the MLP neural network can approximate any continuous nonlinear mapping to any precision, provided there are enough hidden units[4]. In the identification of both Model-I and Model-II nonlinear systems, the MLP neural network are used as a basic structure for identification [1].

Since the basis of using neural network for identification of Model-I and Model-II nonlinear systems is to use the MLP neural network to approximate the nonlinear transfer function, the approximation properties of MLP neural network have significant influence on identification performance. Our simulations show that within the learning range and for relative smooth functions the approximation is good. However outside the learning range the approximation is usually poor. This poor generalization is an intrinsic weakness of a fully connected MLP neural networks [5]. As the mechanism of generalization of MLP neural networks are more like interpolation between known learning sample points [6], the generalization or extrapolation outside the learning range will be poor unless the MLP neural network has a built-in structure which matches the learning problem.

Consider the identification of a Model-I nonlinear system with a MLP neural network. The nonlinear system can be described by a difference equation shown in (2). The identification architecture is depicted in Fig-1.

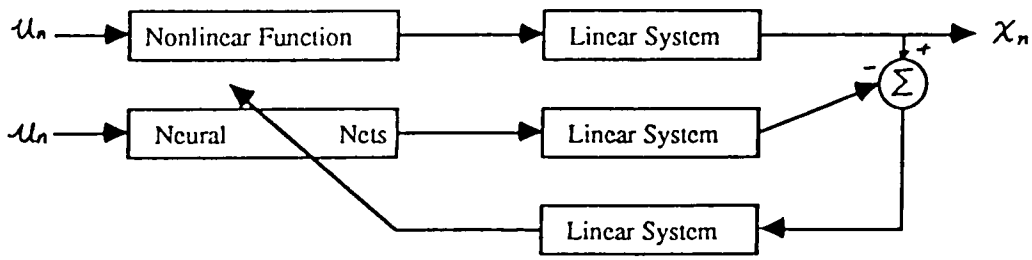


Fig-1

If it is assumed the coefficients a_0, a_1, ..., a_p are known, then the output of the neural network system is

$$\hat{x}_{n+1} = a_0\hat{x}_n + \ldots + a_p\hat{x}_{n-p+1} + NN(u_n) \tag{4}$$

where NN represent the MLP neural network which is used to model the nonlinear function. Let $e_n = x_n - \hat{x}_n$, then from (2) and (4) it obvious that

$$e_{n+1} = a_0e_n + a_1e_{n-1} + \ldots + a_pe_{n-p+1} + f(u_n) - NN(u_n) \tag{5}$$

Let $E_n = f(u_n) - NN(u_n)$, and take the partial derivative of (5) to give

$$\frac{\partial e_{n+1}}{\partial w} = a_0\frac{\partial e_n}{\partial w} + a_1\frac{\partial e_{n-1}}{\partial w} + \ldots + a_p\frac{\partial e_{n-p+1}}{\partial w} + \frac{\partial E_n}{\partial w} \tag{6}$$

$\frac{\partial E_n}{\partial w}$ can be calculated by the back-propagation algorithm, so equation (6) can be used as a revised back-propagation algorithm to update the weights of the neural network which is imbeded in the dynamical system. (see [1] for more detail). Using the above identification scheme, we have found by simulation that the identification of f(u) is good only in the range which is covered by the excitation. Outside this range the identification is poor.

For the Model-II nonlinear system identification, the nonlinear system is described by a nonlinear difference equation shown in (3). If a parallel scheme is used, the neural system which is used to model the plant can be represented as

$$\hat{x}_{n+1} = NN(\hat{x}_n, \hat{x}_{n-1}, \ldots, \hat{x}_{n-p+1}) + u_n \tag{7}$$

where $\hat{x}_n$ is the estimation of x_n, and u_n is the known excitation which is the same as that in (3). From equation (3) and (7), the discrepancy between the plant and the neural system can be calculated as

$$\begin{aligned} e_{n+1} &= x_{n+1} - \hat{x}_{n+1} \\ &= f(x_n, x_{n-1}, \ldots, x_{n-p+1}) - NN(\hat{x}_n, \hat{x}_{n-1}, \ldots, \hat{x}_{n-p+1}) \end{aligned} \tag{8}$$

the e_n is used in the identification processes to adjust the neural network to minimize the discrepancy between the plant and the neural network. If the series-parallel model is used, equation (8) can be replaced by the equation

$$e_{n+1} = f(x_n, x_{n-1}, \ldots, x_{n-p+1}) - NN(x_n, x_{n-1}, \ldots, x_{n-p+1}) \qquad (9)$$

In this paper a series-parallel model is used instead of the parallel model, for which the covergence is difficult to achieve even in linear system identification. The identification architecture is depicted in Fig-2.

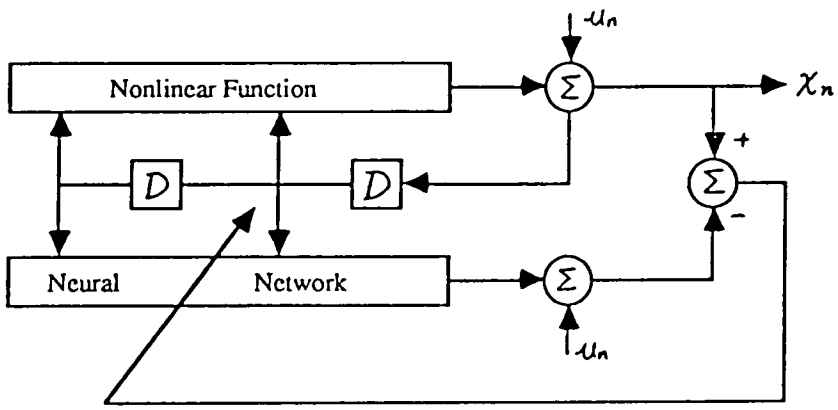


Fig-2

From equation (9) we can see that the identification problem in this case is almost the same as the function fitting problem discussed previously. The difference here is that the samples used for calculating e(k) are determined by the property of the system to be identified. However in the function fitting case the samples can be selected arbitrarily. As mentioned previously the generalization mechanism of the MLP neural network is the interpolation between the learning samples and extrapolation outside the learning region. To make the interpolation match the function which produces the learning samples, the learning samples should be very dense within the learning region of the input space, and generally the extrapolation outside the learning region will be poor, as shown previously. Thus to obtain satisfactory identification, the learning samples should cover the whole input domain which one is interested in, and have sufficient density. The systems and the excitation should meet some demand.

The distribution of state vectors in the state space is called a phase trace or phase portrait. There are several system properties which can influence the phase trace. First, we consider the controllability or reachability of the system. In system theory, controllability means any system state can be reached within finite time with an appropriate excitation[7]. If the system is uncontrollable, for example if the state space consists of several diconnected subspaces, then the identification results will be highly dependant on the initial state. If the initial state lies in a specific subspace, then the identification can only be obtained in this specific subspace.

Apart from controllability, the phase trace is also influenced by the correlation property or bandwidth of the system. Although controllability guarantees that every corner of the state space is reachable under appropriate excitation, the distribution of the phase trace is more influenced by the bandwidth of the system if the excitation is not specifically designed. For example in a highly correlated system the system output is highly correlated when it is excited by a random signal, so the phase portrait is usually restricted to a diagonal region in the state space. Thus the identification would be restricted to this diagonal region as well.

The above discussion about the application of MLP neural networks to nonlinear system identification is restricted to systems which have an asymptotically stable equilibrium point. As is well known there are a large number of nonlinear systems whose attractors are not simply points or limit cycles, but are strange attractors which can lead the system into chaotic behaviour[9],[10]. It could be speculated that because of the special phase traces of strange attractors in state space, the identification of chaotic systems would be poor. On the other hand, for a chaotic system any infinitesimally different starting points will produce significantly different outcomes. So any small modeling error will be amplified to its maximum in the dynamic process. This presents a fundamental difficulty for chaotic system identification.

In conclusion it may be stated that the universality of MLP neural networks does not necessarily provide advantages in applications like system identification. For an MLP neural network without a structure designed to match the system to be identified, the identification results heavily depend on the range of distribution for the phase trace. To obtain satisfactory identification, the system and the excitation should meet some preconditions like controllability, etc. Thus a prior knowledge of the possible system structure is important for satisfactory system identification. However in the case when little structure information is available, the MLP neural network model can always be used as a last resort.

If the identification is only restricted to a small part of the state space, theoretically we cannot say the identification is complete. However from a practical point of view, the results may still have application value. For example, in the narrow band system, correct identification is restricted in the diagonal region, but under general conditions the phase portrait of the system will rarely go out of this range. To drive the phase portrait out of the diagonal region, a strong high frequency excitation is required and would rarely occur in a practical situation. In some cases, although the neural system is a poor model of the real system, it may still be a good predictor. This is especially true of chaotic systems. In the predictor case there is no error accumulation process which is a basic feature of chaotic dynamical systems. Although the above discussion is mainly concentrated on Model-I and Model-II neural network systems, the observations are also applicable to other system identification schemes using neural networks.

All the simulation results will be presented at the Colloquium.

References

[1] Narendra, K.S and Parthasarathy, K *Identification and Control of Dynamical Systems Using Neural Networks* IEEE Transaction on Neural Networks, Vol.1, No.1, March 1990, p4–27

[2] Nauyen, D.H and Widrow, B *Neural Networks for Self-Learning Control Systems* IEEE Control Systems Magazine, April 1990, p18–23

[3] Chu, S.R, Shoureshi, R and Tenorio, M *Neural Networks for System Identification* IEEE Control Systems Magazine, April 1990, p31–35

[4] Funahashi, K.I *On the Approximate Realization of Continuous Mappings by Neural Network* Neural Networks, Vol.2, No.3, 1989, p183–192

[5] Chen, J.R and Mars, P *The Generalization Properties of Neural networks* Internal Technical Report, University of Durham, June 1989.

[6] Broomhead, D.S and Lowe, D *Multivariable Functional Interpolation and Adaptive Networks* Complex System, Vol.2, 1988, p321–355

[7] Kailath,T *Linear Systems* Prentice-Hall, Inc. 1980

[8] Thompson, J.M.T and Stewart, H.B *Nonlinear Dynamics and Chaos* John Wiley & Sons, Ltd. 1986

[9] Ruelle, D *Strange Attractors* The Mathematical Intelligencer 2, 1980 p126-137

Appeared on Proceedings of IJCNN-90-WASH-DC Washington, D.C. Jan.1990

Stepsize Variation Methods for Accelerating the Back-Propagation Algorithm

J.R.Chen and P.Mars
School of Engineering and Applied Science
University of Durham, South Road, Durham, DH1 3LE U.K

Abstract:

In this paper we discuss results on improving the convergence speed of the back-propagation algorithm, and introduce an adaptive stepsize technique and a differential stepsize method to accelerate the convergence speed of the back-propagation algorithm. Simulation results are presented which illustrate the improved convergence.

1.Introduction

The recent revival of research activities in neural networks was significantly influenced by the publication of [1]. With the learning algorithm called the back error-propagation, it was shown that the Multi Layer Perceptron (MLP) can perform interesting computations[1]. Unlike the perceptron analysed by Minsky[2] which can only solve linear separable problems, the MLP , theoretically , can divide the input space into arbitary shape, provided that there are enough hidden units. Thus MLP methods have been applied to several complex pattern classification-like problems, such as that reported in [3]. However the main drawback in applying MLP networks to many real problems is the slow convergence speed of the back-propagation algorithm.

While the back-propagation algorithm is a kind of gradient descent algorithm, error surfaces for learning problems frequently possess some geometric properties that makes the algorithm slow to converge. The stepsize of the algorithm is sensitive to the local shape and curvature of the error surfaces. For example, a small stepsize will make the algorithm take a very long time to cross a long flat slope. On the other hand, a large stepsize will cause the iteration process to bounce between the two opposite sides of a valley rather than following the contour of its bottom. Even if a satisfactory stepsize for one stage of the learning process is found, this does not ensure it will be appropriate for any other stage of the same learning process. On the other hand, the premature saturation of the network units also causes problems for the convergence of the algorithm. Thus in the following we introduce an adaptive stepsize back-propagation algorithm and a simple method for circumventing the premature saturation.

2.Previous Research

There has been some research on improving the convergence speed of the back-propagation algorithm, such as that mentioned in [4][5][6]. In [4] the authors suggested Conjugate gradients, Quasi-Newton algorithm and other more sophisticated algorithms. They are also called second order methods. According to our knowledge, the convergence speed reported in [1] on the XOR problem is the fastest among the existing algorithms. However all these algorithms are much more computationally expensive, especially when the scale of the problem is large, so that in many cases it is impractical to use them. In order to reduce the computation cost of the second order method, a kind of approximation technique has been introduced into the Newton's algorithm[5]. The authors used a diagonal matrix to approximate the Hessian matrix. This makes it possible to derive a back propagation algorithm for the second order derivatives as that for the first order derivatives. But the applicability of this new algorithm depends on how well the diagonal Hessian approximation models the true Hessian[5]. Only when the effects of weights on the output are uncoupled or nearly uncoupled, can the diagonal Hessian represent a good approximation. We have implemented this Newton-like method in our back-propagation simulation program. At this stage we have not found it to exhibit any advantage over the ordinary back-propagation algorithm. This may be due to the use of sub-optimal learning parameters. Just as was mentioned in [5], we found the learning parameters are more critical in obtaining reasonable behaviour with this Newton-like algorithm than with the back-propagation algorithm. Another attempt to use a second order method to improve the convergence property of the back-propagation algorithm was introduced in [6], which is called Quickprop. It uses the difference between two successive $\frac{\partial E}{\partial w}$ as a measure of the change of curvature and uses this information to change the stepsize of the algorithm. E is the output error function, and w represent weights. Using this method a significant improvement on convergence speed has been reported in [6].

In [7] another kind of adaptive stepsize algorithm was introduced. According to this algorithm, if an

update of weights results in reduced total error, the stepsize is increased by a factor $\phi > 1$ for the next iteration. If a step produces a network with a total error more than a few percent above the previous value, all changes to the weights are rejected, the stepsize is reduced by a factor $\beta < 1$, the momentum term is set to zero, and the step is repeated. When a successful step is then taken, the momentum term is reset.

As is well known in adaptive signal processing theory, the direction of the negative gradient vector may not point directly towards the minimum of the error surface. In adaptive filter theory, this kind of bias can be measured by the ratio of the maximum eigenvalue and the minimum eigenvalue of the auto-correlation matrix[8]. Recently an adaptive stepsize algorithm which gives every weight a stepsize which can adapt separately has been proposed[9]. This is only a rough approximation, as it will be noted that these stepsizes adapt on the direction of each weight rather than on the eigenvector direction as required[8][9].

In the back-propagation algorithm, the update of weights can take place after presenting all the training samples to the network or after every presentation of a training sample, they are called batch mode back-propagation and online back-propagation respectively. Generally speaking, online back-propagation algorithms converge faster than the batch mode back-propagation[5][6], and batch mode back-propagation is more likely to fail to converge on a large training sample set[10]. The algorithms described above are all batch mode back-propagation, because for the second order method it can only use batch mode. In the following we introduce an adaptive stepsize online back-propagation algorithm. It is considered to represent an advance on existing algorithms.

3.Adaptive Stepsize Back-Propagation

In designing an appropriate algorithm the following factors should be considered. First the momentum term cannot be set to zero, as the update occurs for every presentation of a new training sample. If the momentum term is set to zero, there exists a risk of losing past experience. Generally speaking. a large training sample set requires a large η value (η is the stepsize for the momentum). This fact has been confirmed by computer simulation[11]. Thus the adaption is restricted to the gradient term. We used the following form of adaptive stepsize algorithm:

$$\alpha(t) = \alpha(t-1)(1 - f(t)\sqrt{E(t)}\ ) \tag{1.a}$$

$$f(t) = u_1 f(t-1) + u_2 \Delta E(t) \tag{1.b}$$

$$\Delta E(t) = E(t) - E(t-1) \tag{1.c}$$

$\alpha(t)$ is the stepsize for the gradient term in the update formula in the back-propagation algorithm. It is the stepsize at moment t. E(t) is the summation of squared discrepencies between the desired output and the actual output at time t. It can be calculated as following:

$$E = \frac{1}{2}\sum_{k=1}^{N}\sum_{i=1}^{p}(d_i^k - o_i^k)^2 \tag{2}$$

$\Delta E(t)$ is the decrement of the E(t). f(t) is a filtered version of $\Delta E(t)$. Actually (1.b) is a first order low-pass recursive filter, which can smooth the significant changes in $\Delta E(t)$, making the algorithm more stable. u_1 and u_2 are the parameters used to control the adaptation. For small u_1 and big u_2, the adaptation is fast, but it is also more likely to be trapped in oscillation. For big u_1 and small u_2, the adaptation is slow, but it is more stable. Thus the parameter selection involves a trade off. In our simulation, we used $u_1 = 0.9$ and $u_2 = 0.3$. The term $(1 - f(t)\sqrt{E(t)}\ )$ also controls the adaptation of the stepsize. If f(t) is positive, that means the tendency of E(t) in the near past is to increase, so $1 - f(t)\sqrt{E(t)} < 1$, the stepsize will be decreased. A similar analysis shows that if the tendency of E(t) is to decrease, the stepsize will be increased. When the E(t) is very small, that is the network has almost learned, the adaption will be very weak, which stablizes the algorithm. The square root is used as compensation, it can amplify the small E(t) to avoid the premature termination of adaptation.

We now present some simulation results to show the advantage of the adaptive step size algorithm. In the diagrams shown, the E defined in (2) are plotted as a function of iteration times for different learning problems. They are called learning curves, and can be used to evaluate the convergence property of the learning algorithm. Their maximum are normalized to 1. In Fig-1 we show comparative simulation results of the non-adaptive back-propagation algorithm and the adaptive algorithm for the 4-4-1 parity problem. It is clear the adaptive stepsize has improved the convergence speed, just as we expected. In our simulation we

find that the improvement on the complex problem are more impressive than that on simple problem. The reason may be that since adaptation is a dynamic process, it needs a finite time to be effective. For simple problems, the learning process is very short, the adaptation process has not sufficient time to play its role. Thus there is only a small effect of adaption on simple learning problems.

4.Differential Stepsize Back-Propagation

Although the adaptive stepsize back-propagation algorithm has improved the learning speed to some degree, it cannot cope with the premature saturation of the network units. We have noted in our simulations that MLP neural nets are often trapped in a very flat valley or so called local minima, in which area the convergence speed is very slow which corresponds to the flat line intervals on the learning curves of Fig-1. This cannot be solved by an adaptive stepsize technique, because the reason for this phenomenon is that the absolute value of weights are growing so fast as to make the units, especially hidden units, prematurely saturated. There is a term like s(1-s) in the update formula for the back-propagation algorithm, in which s is the output state of the unit. It is quite clear that if s is close to 1 or 0, whichever output is desirable, almost no update will be passed backward through that unit. This kind of phenomenon is also known as the flat spot[6]. In [6] the author suggested to change the sigmoid-prime function s(1-s) to s(1-s)+0.1, so it can avoid the flat spot. But according to our simulations, this change often causes the weights to grow so fast as to lead to floating point overflow on the digital computer. Although some weight-decay term may be used to counteract this[6], it makes the algorithm more complex. We have used a very simple method to cope with the flat spot.

A straight forward idea to circumvent the flat spot is to remove the term s(1-s) from the update formula for the output layer, and set the stepsize for the update of weights between the hidden layer and the input layer smaller than that for the weights between the upper layers. We denote the stepsize for the update of weights between the output layer and the hidden layer as α_2, and the stepsize for the update of weights between the hidden layer and the input layer as α_1, then $\alpha_2 > \alpha_1$. We call this the differential stepsize back-propagation algorithm(DSBP). In our simulation, we used $\alpha_1 = 0.1\alpha_2$. The simulation results are shown in Fig-2, and it is very clear the convergence speed is improved considerably.

In [6] the Quickprop algorithm was claimed to be the fastest learning algorithm among the existing algorithms. In order to compare our DSBP with the Quickprop, we have run 30 simulation trials on the 10-5-10 encoder problem. The termination condition for the simulation is that the discrepancy between the desired output and the actual output for every output unit and every training sample is less than 0.1. The average training time for this problem by DSBP is 23.5, with a standard derivation of 3.27. This is only marginally slower than the Quickprop algorithm, for which the average training time is 22.1. However although the Quickprop plus a hyperbolic arctan error function algorithm can reach the same solution with an average training time of 14.01, it is much more complex than DSBP, and a weight-decay term is needed. The results for the simple DSBP algorithm represent a considerable improvement on the standard back-propagation algorithm, which gave an average training time of 129 iterations.

5.Conclusion

From the above discussion, it is clear that the adaptive stepsize technique can improve the convergence speed of the back-propagation algorithm. It is obvious that the degree of improvement for a complex learning problem is greater than that for simple problems. We consider that the potential of the adaptive stepsize technique lies in the area of real large scale application problems, such as Net-Talk[3], in which the training sample set is very big, and the training process may last for a few days. From the simulation results shown above, we can also conclude that the DSBP method we used to circumvent the premature saturation or flat spot is effective. It is also surprising that such a small change to the algorithm can produce such a significant improvement, and confirms the importance of concentrating on a theoretical understanding of the dynamics of the back-propagation algorithm.

References

[1] David E.Rumelhart, James L.McClelland, *Parallel Distributed Processing* Vol-1 The MIT Press 1987

[2] Marvin L.Minsky, Seymour A.Papert, *Perceptrons* Expanded Edition The MIT Press 1988

[3] T.J.Sejnowski, C.R.Rosenberg, *Parallel Networks that Learn to Pronounce English Text* Complex System 1 pp145-168 1987

[4] A.R.Webb, David Lowe, M.D.Bedworth, *A Comparison of Nonlinear Optimisation Strategies for Feed-Forward Adaptive Layered Network* Royal Signals and Radar Establishment Memorandum 4157 1988

[5] Sue Becker, Yann le Cun, *Improving the Convergence of Back-Propagation Learning with Second Order Methods* Proc 1988 Summer School on Connectionist Model, Carnege-mellon Univ, pp29-37

[6] Scott E.Fahlman, *Fast-Learning Variations on Back-Propagation : An Empirical Study* Proc 1988 Summer School on Connectionist Model, Carnege-Mellon Univ, pp38-51

[7] T.P.Vogl, J.K.Mangis, A.K.Rigler, W.T.Zink and D.L.Alkon, *Accelerating the Convergence of the Back-Propagation Method* Biological Cybernetics vol-59 pp257-263 1988

[8] S.Haykin, *Adaptive Filter Theory* Englewood Cliffs, NJ, Prentice-Hall 1986

[9] Robert A.Jacobs, *Increased Rates of Convergence Through Learning Rate Adaptation* Neural Network vol-1 pp195-307 1988

[10] P.J.Lloyd, *Guidlines for Implementing Problems on a Multi-layer Perceptron Network* RIPRREP/1000/27/88

[11] G.Tesauro and B.Janssens, *Scaling Relationships in Back Propagation Learning* Complex System vol-2 No.1 pp38-44 1988

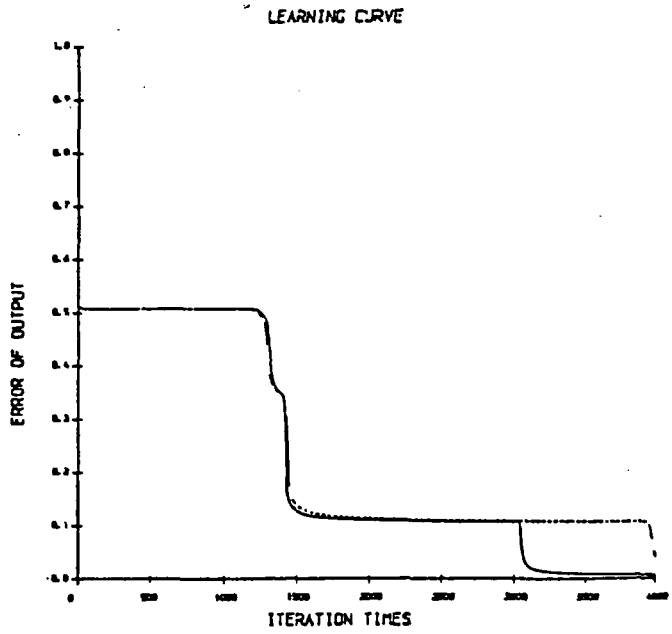


Fig-1

Learning curves for the 4-4-1 parity problem. Broken line stands for the learning curve of non-adaptive algorithm. The initial random value of weights are within the range(-0.5, 0.5), w-seed=5697, th-seed=8461, $\alpha = 0.4$, $\eta = 0.9$.

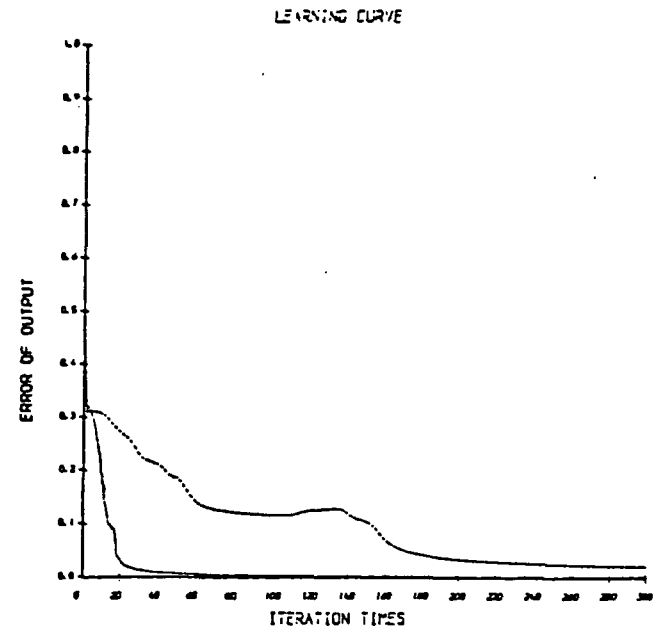


Fig-2

Learning curves of the 10-5-10 encoder problem. Solid line stands for the learning curve of the differential step-size back propagation algorithm(DSBP). The initial random value of weights are within the range(-1, 1), w-seed=4581, th-swwd=818953, $\alpha = 0.6$, $\eta = 0.9$. But for the DSBP $\alpha_1 = 0.06$, $\alpha_2 = 0.6$.