KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT TOEGEPASTE WETENSCHAPPEN
DEPARTEMENT ELECTROTECHNIEK
AFDELING ESAT
Kardinaal Mercierlaan 94 — 3001 Leuven (Heverlee)

DESIGN OF NEURAL NETWORKS WITH APPLICATIONS TO ASSOCIATIVE MEMORIES, IDENTIFICATION AND CONTROL

Jury :
Chairman : Prof. W. Dutré, vice-dean
Prof. J. Vandewalle, promotor
Prof. B. De Moor, promotor
Prof. A. Barbé
Prof. D. Bollé
Prof. H. Van Brussel
Prof. D. Van Compernolle

Thesis submitted to obtain the degree of doctor in the applied sciences

by

Jianbin HAO

UDC : 681.5.021

September 1994

Preface

Although it is impossible to mention all the people who have provided their support and encouragement during the course of this research, I would like to record my thanks to the following people.

First of all, I would like to express my sincere gratitude and appreciation to my promotor Prof. J. Vandewalle, who provides me with the opportunity to study in ESAT, K.U. Leuven, for his constant supervision, stimulation and encouragement. His profound knowledge and broad research interests contributed a great deal to this work. As the head of ESAT-SISTA, he is responsible for the healthy research atmosphere established within the group, which is very important to the accomplishment of this work.

With a great pleasure I wish to thank my co-promotor Prof. B. De Moor, for his insight comments and suggestions on my work and for the interesting discussions I had with him. His enthusiasm and creativity keep us all going in SISTA.

I am heavily indebted and extremely grateful to my past daily work adviser Dr. Shaohua Tan, for his introducing me to the field of neural networks, for his continuous stimulation, for his invaluable support and guidance and for many enlightening discussions. His contributions to the accomplishment of this work have been partially reflected in the thesis.

Many thanks go to the members of the reading committee, Prof. J. Vandewalle, Prof. B. De Moor, Prof. A. Barbé and Prof. D. Bollé, for their critical review and constructive comments which made this thesis more readable. Thanks also go to the other members in the jury, Prof. W. Dutré, Prof. H. Van Brussel and Prof. D. Van Compernolle.

I wish to thank my colleagues in ESAT-SISTA, both past and present, for

the help and assistance they provided in various way. In particular, I would like to thank our secretaries Ms. I. Tokka and Ms. A. Deforce. They are always kind and ready to provide their help.

Last but not least, I would like to thank my wife Hua, for her love and encouragement. Her support and patience have been of vital importance to the accomplishment of this work.

Jianbin Hao
September 1994

Abstract

This dissertation treats the issues of design and training of neural networks for the problems of classification, memory association and system identification and control.

Firstly, for the two classes (with binary input patterns) classification problem, or dichotomy problem, a general approach based on the geometric interpretation to hidden layer of a multilayer feedforward network is developed.

Secondly, aiming at improving the performance of a standard Hopfield associative memory, a fixed point programming method is first derived, then an extended Hopfield model of associative memories with hidden structure is devised, and finally a feedforward model of associative memories which is much simpler and more powerful is proposed.

Thirdly, making use of the unique learning ability inherent in the neural networks, we have designed a rule based neural controller for the cart-pendulum system, which is considered to be one of the standard benchmark nonlinear control problems. Further, based on the important universal approximation property of the feedforward neural networks, both multilayer feedforward neural networks and radial basis function networks are designed and trained for nonlinear system identification. New training algorithms derived from the well established linear adaptive system theories and the improved modeling structures are proposed.

Contents

1 Introduction **1**
- 1.1 History of artificial neural networks 1
- 1.2 Recurrent networks versus feedforward networks 3
- 1.3 Nonlinear systems versus linear techniques 4
- 1.4 Chapter by chapter overview 5
  - 1.4.1 Geometric approach 6
  - 1.4.2 Hopfield associative memories 6
  - 1.4.3 Identification and control 8

2 Feedforward Networks Design: Geometric Approach **10**
- 2.1 Introduction 11
- 2.2 The n-parity problem 12
- 2.3 Mapping an arbitrary dichotomy to an n-parity problem 16
- 2.4 An example 21
- 2.5 Discussions and conclusions 25

3 Recursive Network Design: Associative Memories **27**
- 3.1 Introduction 28
  - 3.1.1 Associative memory 28
  - 3.1.2 The Hopfield model of associative memories 32
  - 3.1.3 Outline of this chapter 38
- 3.2 Fixed points programming by error back propagation 38
  - 3.2.1 Introduction 39
  - 3.2.2 Discrete Hopfield nets and feed-forward nets 40
  - 3.2.3 Design of Hopfield nets by solving inequalities 42

3.2.4 Design examples 47
3.2.5 Conclusions 49
3.3 Pattern storage and Hopfield nets with hidden structure 49
3.3.1 Introduction 50
3.3.2 Expanded Hopfield nets 51
3.3.3 Design rule 54
3.3.4 Relations between attracting basins and the network structure 60
3.3.5 Conclusions 62
3.4 An optimization based design rule 63
3.4.1 Introduction 63
3.4.2 Definitions and conventions 64
3.4.3 The new design rule 69
3.4.4 Illustrative examples 72
3.4.5 Conclusions 80
3.5 CNN and associative memories 81
3.5.1 Introduction 81
3.5.2 Discrete-time cellular associative memory 82
3.5.3 Design rule and pattern conditions 84
3.5.4 A simple example 88
3.5.5 Conclusions 89
3.6 Conclusions and discussions 89

4 Pattern Association and Feedforward Model of Associative Memories 92
4.1 Introduction 93
4.2 Neurons, multi-layer feedforward networks and vector associations 95
4.3 Pattern association by two-layer feedforward nets 98
4.4 Association network as hetero-associative memory 101
4.5 Design examples 103
4.6 Conclusions 105

5 Rule-Based Neural Controller for the Inverted Pendulum System 106
5.1 Introduction 107
5.2 The Plant dynamics and problem decomposition 111

5.3 Rule extraction and neural net implementation 113
5.3.1 Neural net implementation 114
5.3.2 Pendulum sub-controller 115
5.3.3 Cart sub-controller 116
5.3.4 Switching sub-controller 117
5.4 Simulation 120
5.5 Discussions and conclusions 120

6 Predictive Control of Nonlinear Systems by Feedforward Networks 124
6.1 Introduction 124
6.2 Multilayer feedforward neural networks and back propagation algorithm 127
6.3 Identification and control 130
6.3.1 Identifications 131
6.3.2 Predictive control 132
6.4 Simulation results 136
6.5 Discussions and conclusions 144

7 Multi-Channel Signal Modeling by RBF Neural Networks 145
7.1 Introduction 145
7.2 The modeling scheme 150
7.2.1 General idea 150
7.2.2 Weight updating rule 152
7.2.3 Convergence analysis 155
7.2.4 Additional remarks on the weight updating rule 157
7.3 Determination of RBF neural net structure 158
7.3.1 Heuristic methods and their drawbacks 159
7.3.2 Spatial Fourier theory based structure determination . . 160
7.4 Simulation analysis 165
7.5 Discussions and conclusions 175

8 General Conclusions 177
8.1 Direct design method for MLP 177
8.2 Neural associative memories 179
8.3 Nonlinear system identification and control 182
8.4 General conclusions 185

List of Figures

2.1 A simple example of constructing two parallel hyperplanes from one hyperplane in 3-dimensional space. 13
2.2 The Hard Limiter processing function for neurons and a neuron with n inputs and one output. 14
2.3 One-hidden-layer perceptron with n hidden neurons and one output neuron (with hard limiter function) thresholding at 0.5 solving the n-parity problem in an n-dimensional hypercube. . 15
2.4 The neural net for a dichotomy problem in 3-dimensional hypercube. 25

3.1 An address addressable memory with each location storing one pattern. With the eight address lines shown in the figure, there could be up to 256 locations to store patterns, while only 40 of them are shown here. 29
3.2 An associative memory with each element storing a portion of multi patterns and each pattern is stored over multi elements. . 30
3.3 The configuration of the Hopfield net. The thresholding function for each neuron is assumed to be the Sign function. 32
3.4 The nonlinear function of the neurons and the special feedforward neural net structure used in the approach. Note that there are direct connections from the input nodes to the output neurons. 41
3.5 Reformulation of a consistent set of inequalities into a feedforward net. The final weights after training this net with the back-propagation algorithm will be the solution to the inequalities. 44

3.6 The n-neuron Hopfield net for a consistent set of data. 44
3.7 The Hopfield net for an inconsistent set of data. $O_1, O_2, \cdots, O_n$ are standard neurons while $H_1, H_2, \cdots, H_m$ are hidden neurons. Same as in Figure 3.6, these neurons are connected via weighted links, while in this case, the symbols for the weights and the directions of connections are dropped for the clarity of the figure. 46
3.8 The scheme of the extended Hopfield net 53
3.9 An illustration of the pattern condition: (a). A legitimate pattern specification; (b). A contradictory specification. 55
3.10 Three different types of attracting basins and fixed points. . . . 65
3.11 The different convergent behaviors within convergence ball and strict convergence ball. 66
3.12 Radii of the *strict convergence ball* of 32-neuron Hopfield net obtained with different methods. "x" denotes that not all of the given patterns can be made fixed points by Hebbian rule. . 76
3.13 Radii of the *strict convergence ball* of 64-neuron Hopfield net obtained with different methods. "x" denotes that not all of the given patterns can be made fixed points by Hebbian rule. . 77
3.14 Radii of the *strict convergence ball* of 128-neuron Hopfield net obtained with different methods. "x" denotes that not all of the given patterns can be made fixed points by Hebbian rule. 78
3.15 Number of patterns that can be stored with an attraction basin of at least unit Hamming distance plotted as a function of number of neurons. 79
3.16 The configuration of the discrete cellular neural network. . . . 83
3.17 The typical connections for each neuron in a cellular neural net. 85
3.18 Pattern P_1, P_2 and P_3 have been stored in a CNN, when $\tilde{P}_i$ is presented as input, the CNN will output P_i $(i = 1,2,3)$. 86
3.19 One possible modification of the standard cellular neural net . 89

4.1 An example of the two-layer feedforward net. The output neurons are sign neurons and the hidden neurons are hard-logic neurons. 96

5.1 The scheme of the rule based controller design. 108

5.2 The cart-pendulum system, the angle and angle speed are positive in clockwise direction. 110
5.3 The overall control scheme. 113
5.4 Geometric implementation of neural network. (a) Rule in geometric form. (b) Neural network implementation. 114
5.5 The Pendulum Sub-Controller. 116
5.6 The Cart Sub-Controller. 117
5.7 The whole neural controller with SSC being highlighted by dotted lines. The symbol [d] represents a unit time delay. . . . 118
5.8 The result of simulation (pendulum). (a) shows the whole evolution process of angular displacement against the time samples. (b) is a zoom-in picture of part of (a) from which one can clearly see that the pendulum moves closely around the zero point irregularly after stabilization. (c) shows the whole simulation process of angular speed against the time samples. (d) is a zoom-in picture of part of (c) which indicates the pendulum keeps swinging back and forth after stabilization. 119
5.9 The first 500 samples of simulation (pendulum). The discontinuity of the curve in (a) is because that we have chosen the range of the angular displacements to be $[-\pi, \pi]$ and these two values should be looked upon as the same position for the pendulum. 121
5.10 The result of simulation (cart). (a) shows the displacement of cart against time samples for the whole simulation process. (b) is a zoom-in picture of part of (a) in which it is clearly shown that the cart moves irregularly within a small range of the origin of the track. (c) shows the speed of cart evolves against time samples. (d) shows a detailed picture of (c) from which one can see that the cart keeps moving towards or away from the origin of the track. 122

6.1 The structure of the neural net used in this work. 128
6.2 Direct identification. (a) Parallel model. (b) Series-parallel model. 132
6.3 Inverse identification. 133

6.4 Optimization based predictive control scheme. [D] stands for multiple delay units and the wide line stands for multi signal flows. The initial set up of NN1 comes from the off-line identification and is further tuned by the 'error' signal, while NN2 is always a copy of NN1. 135
6.5 Simulation result of Example 1. (a) Result of off-line identification, solid line: desired output, dashed line: real output, dotted line: error. (b) Result of control, solid line: desired output, dashed line: real output, dotted line: control input. . . 138
6.6 Simulation result of Example 1 when the plant was modified after 150 iterations, solid line: desired output, dashed line: real output, dotted line: control input. 139
6.7 Simulation result of Example 2. The upper figure shows the result of control on the unmodified plant. The lower figure shows the result of control on a plant modified after 150 steps. Solid line: desired output, dashed line: real output, dotted line: control input. 141
6.8 Simulation result of Example 3. The upper figure shows the result of control on the unmodified plant. The lower figure shows the result of control on a plant modified after 150 steps. Solid line: desired output, dashed line: real output, dotted line: control input. 142
6.9 Simulation result of Example 4. (a) Result of off-line identification after 100000 iterations, solid line: desired output, dashed line: real output, dotted line: error. (b) Result of predictive control, solid line: desired output, dashed line: real output, dotted line: control input. 143

7.1 The structure of a RBF neural net. 148
7.2 A simulation plot of mean-square modeling error versus the width σ. Solid line: A RBF net with 42 hidden neurons; dashed line: A RBF net with 13 hidden neurons. 162
7.3 The grid showing the locations of the centers in *Example 1*. The operating region $\mathcal{S}$ is the rectangular $[-1.8, 1.8] \times [-1.1, 1.1]$. Here Δ (the horizontal or vertical distance between two neighboring centers) is chosen to be 0.31. 166

7.4 The convergence of the prediction error in *Example 1*. 168
7.5 The result of modeling in *Example 1*. Solid line: Real signal; dashed line: Modeled signal; dotted line: Modeling error. . . . 169
7.6 The convergence of the mean-square prediction error in *Example 2*. 171
7.7 The result of modeling in *Example 2*. Solid line: Real signal; dashed line: Modeled signal; dotted line: Modeling error. . . . 172
7.8 The result of modeling in *Example 3*. (a) is for the first channel and (b) is for the second channel signal. Solid line: Real signal; dotted line: Modeled signal. 174

Chapter 1

Introduction

In this chapter, we will first give a brief overview of the development history of artificial neural networks and the problems dealt with in this dissertation. Then we will discuss recurrent networks versus feedforward networks with applications to associative memories. We will also discuss the possibilities of some linear techniques being used for the weight updating of Radial Basis Function neural networks. Finally we will give a chapter by chapter overview of the dissertation.

1.1 History of artificial neural networks

The concept of artificial neural networks (or artificial neurons) was first proposed by McCulloch and Pitts in 1943 in [58]. In that paper they outlined the first formal model of an elementary computing neuron. The model included all necessary elements to perform logic operations, and thus it could function as an arithmetic-logic computing element. Since then, although the research activities in the field of artificial neural networks had never ceased, there was not a real break-through both in the number of researchers in this field and the results published for almost forty years.

During the past decade or so, artificial neural network as a revitalized research field has received a tremendous attention and made huge progress both in the theoretical and applicational aspects. Researchers in the areas ranging from engineering, computer science, physics to biology, psychology

and economics have initiated a great number of research programs dealing with neural networks and their applications. Many fundamental results which are crucial to the development of the field have been obtained in this period. The starting of this renaissance era is marked by the publication of the two famous papers by John Hopfield [42][43] on the neural network structures of associative memories. Another landmark in the history of artificial neural networks is the publication of a two volumes book on parallel distributed processing, by the PDP group in U.C. San Diego in 1986 [57]. The new learning rules and other concepts introduced in this work have removed one of the most essential network training barriers that grounded the mainstream efforts of the mid-1960s.

Since 1986-87, many new research programs were initiated in the field and many new teams started their works on neural networks. The intensity of research in the neurocomputing discipline can be measured by the quickly growing number of conferences and journals devoted to the field. The list of applications that can be solved by neural networks has expanded from small test-size problems and toyish examples to large practical tasks. Quite a few very-large-scale integrated neural network chips have been fabricated. Although neurocomputing does already have an interesting history, the field is still in its early stages of development. Many important theoretical problems have not been solved and a lot of potential application areas for neural networks are yet to be explored.

In this dissertation, three problems will be addressed. First we present a geometric approach to the structural synthesis of multilayer feedforward neural networks. Then we analyze neural network models of associative memories and finally we discuss nonlinear system identification and control by neural networks. The approaches to be used will be a combination of some existing techniques and improved or novel methods. More specifically, we will discuss neural associative memories with the traditional recurrent network structure and a new structure of feedforward network models. Needless to say, any neural network is a typical nonlinear system. The determination of its structure and weights are, for the most cases, very complicated nonlinear processes. In the latter part of the dissertation, we will discuss a method which takes advantage of well established linear adaptive control theories to guarantee the convergence of the weight updating process for the Radial Basis Function (RBF) neural

networks.

1.2 Recurrent networks versus feedforward networks

The Hopfield neural network is a typical example of recurrent networks, in which neurons[1] are partially or fully connected to each other and the outputs of part or all of the neurons are fed into other neurons or themselves as inputs. For this type of networks, the final outputs of all the neurons will be the stable state of the neural network. One of the successful application examples of the Hopfield neural network is its being used as a model of associative memories, where the stored patterns are recalled via associations instead of addresses.

The reason that recurrent neural networks can be used as associative memories is that there exists a convergent dynamical process in a class of recurrent neural networks. Using certain weight setting up algorithms, the patterns required to be stored can be enforced as stable equilibria of the dynamical process of such kind of neural networks, which then can be recalled by activating the network dynamics by a proper set of initial states [42][43]. The performance of such a neural associative memory is measured by the global network stabilities as well as the local stable equilibria and their attracting basins which are determined for the most part by the connecting weights.

The Hopfield associative memories guarantee the global network stabilities by superimposing the symmetric structure on the weight matrices, while the weight matrix elements are calculated using some learning algorithms, e.g., the Hebbian rule [42][43]. The Hopfield associative memories designed this way can only meet the requirements on the equilibria if the patterns to be stored are well separated (preferably, orthogonal to each other), while the problem of the attracting basins is not addressed at all [42][43]. Aiming at solving such kind of problems, we proposed several methods to improve the performance of the neural associative memories based on a sometimes modified Hopfield network structure, which is still a recurrent one.

[1] The term "neuron" used in this dissertation refers to certain computational unit with precise definition of input-output transfer function. It does not intend to suggest any connection with biological neurons. This latter statement applies to the term "neural networks" as well. Thus, while the same structures may be explored elsewhere for possible links with the biological neural networks, our interest in them is directed purely to their mathematical nature.

Although the modified Hopfield associative memories outperform the original model in some aspects, it is shown that there are still many things need to be done to further improve the performance, especially about the attracting basins. Based on the past experiences with feedforward networks and the insights to the recurrent neural associative memories, we propose a feedforward model of neural associative memories. First of all, instead of using a convergent dynamical process as the retrieval mechanism, we use a feedforward neural network as the mapping system to recall the stored patterns from the input patterns. Second and most importantly, by defining the attracting basins as hyper-balls centered around each of the stable equilibria and taking the sizes of these hyper-balls as design parameters in the feedforward model, the associative memories are guaranteed to be able to store the given patterns with the preassigned size of attracting basins. This feedforward model of associative memories can be considered to be a successful example of using feedforward networks to solve problems in the fields traditionally dominated by the recurrent networks.

1.3 Nonlinear systems versus linear techniques

Artificial neural networks are typically nonlinear systems composed of interconnected elements (neurons) with nonlinear processing functions. Although the nonlinearities of each of the processing elements of a network are normally very simple (sometimes just a piecewise linear function), the overall processing function of the whole network could be very complicated and powerful. Owing to their nonlinear nature and learning abilities, neural networks are especially good at nonlinear function approximation, which can be extensively used in nonlinear system identification and control, time series prediction, complex multi-class classification and self-organization.

Traditional methods for designing feedforward neural networks are more or less in the following trend: First determine the number of layers and the number of neurons in each layer, then use some learning algorithms (mostly gradient descent method) and a set of sample data to train the network, in the hope that at the end of the learning process, the network will perform as desired. Although this kind of methods work well for quite a few problems, there are several drawbacks associated with the approach:

1. There is a lack of theoretical proof for most of the algorithms that they will converge to the global minima, mostly, they will end up in local minima.

2. The convergent processes are normally too long to fine tune the algorithms.

3. Because of the high complexities of the algorithms, it is very difficult if not impossible to analyze the properties of the algorithms, such as, convergence speed, error bound, etc.

In the third part of this dissertation, we will introduce a novel learning algorithm for the training of radial basis function (RBF) neural networks. The most significant feature of this algorithm is that by making use of some well established concepts and results in linear and nonlinear system theories, e.g., state space model, linear adaptive systems, error models and Lyapunov methods, a fast convergence of the algorithm is guaranteed. The idea is to first embed the prediction error (the error between the network output and the desired output) and an auxiliary error in a linear error model [55] (with a nonlinear feedback loop). Then the linear system is proved to be asymptotically stable using a Lyapunov function method. Due to the specially chosen Lyapunov function, on one hand, the linear system will settle down at the point where the prediction error is zero, on the other hand, weight updating will end up with the weight error (error between the ideal weight and current weight) being zero as well.

1.4 Chapter by chapter overview

The contents of this dissertation can be grouped into three parts: I. A geometric approach to the structural synthesis of multilayer feedforward neural networks (Chapter 2); II. Improvements of the Hopfield model of associative memories and feedforward models of neural associative memories (Chapter 3 and Chapter 4); III. Signal modeling and identification and control of nonlinear dynamical systems by multilayer feedforward neural networks and RBF neural networks (Chapter 5, Chapter 6 and Chapter 7). In this section we will give an overview of each of these chapters.

1.4.1 Geometric approach

Chapter 2 presents a general approach to the synthesis of multilayer feedforward networks used for the classification of binary inputs in two classes. The underlying idea is to map an arbitrary two classes classification problem to the so-called n-parity problem and the latter can then be solved by applying a special structure. The approach leads to a two-hidden-layer network, and each of the hidden layers has a clear geometrical interpretation. Our construction also provides some insight into the processing nature of multilayer feedforward networks.

Different from those learning based methods, in which the structure of the network (number of hidden layers and number of neurons in each layer) has to be determined in an *ad hoc* basis before the learning starts, the geometric approach determines the structure and the weights altogether and the number of neurons needed is normally less than those of the other methods.

This chapter is based on [26].

1.4.2 Hopfield associative memories

As one of the first neural network models proposed and studied by the researchers, the Hopfield neural network is undoubtedly very important both from a theoretical and a practical point of view. Due to the convergent dynamics and the potential implementability of large number of neurons, Hopfield nets are mainly designed as associative memories. Although this model of neural associative memories has been investigated extensively since 1982 [42], two main problems, i.e., storage capability and capacity, are far from solved. Many efforts have been made either to improve the design rule for the Hopfield model or construct new models of neural associative memories to solve the problems. This part of the dissertation represents such an effort.

Starting from the basic structure of the Hopfield net, we make in **Chapter 3** a series of improvement to its design rule – Hebbian rule – and propose a modified Hopfield net for the purpose of associative memory. First of all, we describe a new systematic approach to design discrete Hopfield associative memories. The underlying idea is to translate such a design problem into solving a set of inequalities. Along with the consideration of the consistency, this latter problem is further formulated into determining the weights of a

feed-forward net, which can then be solved by applying the well-known back-propagation method.

Secondly[2] a new formulation of neural associative memories is proposed. Centered around the fundamental issue of memory storage, we examine the deficiencies associated with the standard Hopfield net. To overcome the problems, we pursue a data-driven design approach by modifying the configuration of the Hopfield net to allow hidden structures. As important results, we show how the well-known sum-of-outer product rule can be utilized to explore the freedom provided by the hidden structures leading to the desired memory performance.

Thirdly, a new method to construct the weight matrix for the discrete Hopfield associative memory is presented. The most important feature of the method is the explicit introduction of the size of attracting basins to be a main design parameter, and the weight matrix is obtained as a result of optimizing such a parameter. Another feature is that all the connection weights can only assume three different values, -1, $+1$ and 0, which facilitates the VLSI implementation of the weights. Compared to the widely used Hebbian rule, the method can guarantee that all the given patterns are stored at least as fixed points, regardless of the internal structure of the patterns. The new design method is illustrated by a few examples, and the performance of the obtained networks is compared favorably to those obtained using the Hebbian rule.

As an extra result, in this chapter the problem of designing cellular neural networks in order to function as associative memories is also tackled. We consider a discrete time version of the cellular neural nets [51] featuring simple thresholding neurons and the synchronous state-updating rule and the Hebbian rule is adopted as the memory design rule. Important issues, such as the memory capacity and the size of attracting basins are discussed.

This chapter is based on [27][35][90][92][93].

Chapter 4 tries to tackle the problem of associative memories from a different viewpoint, i.e., pattern association. Indeed, the retrieval process of a target pattern (stored memory) from an input pattern is nothing more than the association between these two patterns. The difference is that a retrieval process is implemented by a convergent dynamics in the Hopfield net, while

[2]The idea of introducing a hidden structure to standard Hopfield nets originated from Dr. Shaohua Tan.

a pattern association can be realized through static mapping. Suppose an arbitrary set of bipolar pattern[3] associations $\{p_i \longrightarrow q_i\}$, $i = 1, 2, \ldots, k$ are given, the underlying idea of the technique is as follows: First use a layer of the hard-logic neurons (neurons with processing function's output being 1 or 0, depending on whether the input is larger or smaller than 0) to identify which of those p_i's is presented as input pattern in the *winner-take-all* [107] fashion. Then a second layer of the so-called sign-neurons (neurons with processing function's output being 1 or -1, depending on whether the input is larger or smaller than 0) is used to pick up the corresponding pattern q_i. An important feature of this two-layer network is that it can be used as an error-correcting associative memory if the thresholds of the hard-logic neurons in the first layer are properly adjusted. Another advantage of the technique is that neither extensive computation nor heuristics are involved in the determination of the network size, the connecting weights and the threshold values, making it possible to build feedforward networks for association problems of extremely large size.

This chapter is based on [28][29][34][36].

1.4.3 Identification and control

This part of the dissertation is devoted to the modeling of multichannel signals and the identification and control of nonlinear dynamical systems by feedforward networks. **Chapter 5** describes a rule-based neural controller for the cart-inverted pendulum system. The control task is to swing up a pendulum mounted on a cart from its stable position (vertically down) to the zero state (up right) and keep it there by applying a sequence of two opposite constant forces of equal magnitude to the center gravity of the cart. In addition, the displacement of the cart itself is confined to within a pre-set limit during the swinging up action, and it will eventually be brought to the origin of the track. This is truly a nontrivial nonlinear regulation problem, and is considerably more difficult compared to the pendulum balancing problem (and its variations) widely adopted as a benchmark test system for neural controllers. Through the solution of this specific control problem, we try to illustrate a heuristic neural

[3]Bipolar patterns are vectors whose entries are either -1 or 1. Unless otherwise stated, all the vectors are assumed to be column vectors.

control approach with task decomposition, control rule extraction and neural net rule implementation as its basic elements.

This chapter is based on [30][33][91].

Chapter 6 deals with the problem of identification and control of general nonlinear dynamical systems. By using the properties of universal approximation of multilayer feedforward neural networks, a class of discrete nonlinear dynamical systems are modeled by a network with two hidden layers. The backpropagation algorithm is then used to train the model to identify a nonlinear system up to within a desired degree of accuracy. Based on the identified model, a one-step-ahead predictive control scheme is proposed in which the future control inputs are obtained through some nonlinear optimization process. Making use of the on-line learning properties of neural networks, the predictive control scheme is further developed into an adaptive one which is robust to the incompleteness of identification.

This chapter is based on [31][32][37].

Chapter 7[4] presents a neural network based recursive modeling scheme that constructs a nonlinear dynamical model for a given discrete multi-channel signal and the corresponding excitation. Using so-called Radial-Basis-Function (RBF) neural nets as generic discrete-time nonlinear model structure and the ideas developed in classical adaptive control theories, we have been able to derive a stable and efficient weight updating algorithm that guarantees the convergence for both the prediction error and the weight error. Elements of the spatial Fourier transform and sampling theory have been employed to provide the guidelines in choosing the parameters associated with the network structure and the RBF neurons. A simulation analysis is also carried out to highlight the practical considerations in using the scheme.

This chapter is based on [94][95][96][97][98].

[4] In this chapter, the idea of using classical adaptive control theories to build up the weight updating rule and using a Lyapunov function method to prove the convergence of the algorithm originated from Dr. Shaohua Tan and some of the results in this chapter is the outcome of a joint work between the author and him.

Chapter 2

Feedforward Networks Design: Geometric Approach

In this chapter, we will discuss the problem of designing a multilayer feedforward neural network for two-class classifications with binary inputs. Through a geometric interpretation of the functions of hidden layers in a multilayer feedforward network, we will built up a network composed of two functional parts to solve an arbitrarily given dichotomy problem. The first part of the network is a mapping mechanism which is used to map an arbitrary dichotomy problem to an n-parity problem, while the second part is a standard network structure for solving n-parity problems.

The motivation for studying the geometric approach to dichotomy problems in this chapter are two folds: On one hand, it is to provide a direct design method (which leads to the geometrical determination of the structure of the neural net, i.e., its layers, weights and thresholds) as an alternative method to those learning based methods; On the other hand, it is to interpret the functionality of the hidden layers in a neural net from a geometrical point of view. The functionality of the hidden layer in this case can be regarded as mapping various, much more difficult (if not impossible) classification problems in a lower dimensions to standard (here they are n-parity problems), much easier classification problems in a higher dimension.

The design of feedforward neural networks, such as, multilayer perceptrons, is usually accomplished in two steps: First, some priori knowledge or *ad*

hoc results are involved to determine the structure of the network, i.e., number of layers and number of neurons in each layer. Second, the network is trained with massive input-output samples via some algorithm, e.g., back-propagation algorithm, in the hope that the learning algorithm will converge to an appropriate weight matrix. In this chapter we will provide a geometric method as an alternative approach.

2.1 Introduction

Designing a multilayer perceptron for the general purpose classification has important practical implications. Since the capability of multilayer perceptrons to realize arbitrary dichotomies (or two-class classifications) is limited, *i.e.,* not all perceptron structures are capable of implementing all the dichotomies defined among 2^m different binary inputs in an m-dimensional hypercube [106], the most important step in a design procedure is the determination of the number of the layers and the amount of nodes in each layer apart from the determination of the weights and the threshold values. Unfortunately, there is no general principle or guideline available at this moment for such a synthesis task. Hence normal design often proceeds on an *ad hoc* and empirical basis. The methods generally lead to structures which only deal with a particular classification problem [78][83].

Some efforts have been made recently to understand the structural nature of the multilayer perceptron, and a number of methods have been developed trying to access the type of classification ability and to come up with a rough estimation of the hidden units [6][63]. These methods, however, are only valid for some restricted input patterns (e.g., the inputs which are in the so-called "general position" [6], etc.), and not amenable to the general case. There are also some design procedures to cope with the general case with the number of the nodes in the net being $N-1$, $N/2$, etc., where N is the number of input patterns to be classified. The structures produced by these methods, however, have too many nodes, and thus are considered to be impractical.

2.2 The n-parity problem

In this section, after defining an n-parity classification problem, we develop a geometrical method to design a one-hidden-layer neural network with $n+1$ neurons to solve this problem. This one-hidden-layer network will be the second part of the whole network for solving an arbitrary dichotomy problem.

Let the vertices of an n-dimensional hypercube be denoted by n-dimensional vectors $\{-1,1\}^n$. Let the *even vertices* be defined as those vertices with even number of 1's as their elements and all the rest are -1's, and let the *odd vertices* be defined as those vertices with odd number of 1's as their elements and all the rest are -1's.

Definition 1 *An n-parity classification problem is to classify the odd vertices and the even vertices among all the 2^n vertices into two different classes.*

Clearly, there could be many methods to solve this n-parity problem, whereas here we will provide a neural network solution to the problem with geometrical interpretation. To begin, note that in an n-dimensional hypercube it is always possible to find $n+1$ different hyperplanes of dimension $n-1$ such that each of these $n+1$ hyperplanes contains and only contains those vertices having the same number of 1's as their elements. More precisely, since any $(n-1)$-dimensional hyperplane in an n-dimensional space can be represented by

$$a_1x_1 + a_2x_2 + \cdots + a_{n-1}x_{n-1} + a_nx_n = b, \tag{2.1}$$

where a_i $(i = 1,2,\ldots,n)$ and b are constant coefficients, x_i $(i = 1,2,\ldots,n)$ are variables, the aforementioned $n+1$ hyperplanes can easily be constructed by setting $a_1 = a_2 = \cdots = a_n = 1$ and $b = 2k - n$, or written explicitly

$$x_1 + x_2 + \cdots + x_n = 2k - n \quad (k = 0,1,\ldots,n) \tag{2.2}$$

where k is the number of 1's among the vertices' elements. It can easily be verified that only the vertices with a number k of 1's among their elements lie on (2.2) and the vertices with a different number of 1's do not lie on it. With the above construction and by setting k to be $1,\ 3,\ 5,\ldots,(2\lceil n/2\rceil - 1)$ we can select p $(p = \lceil n/2\rceil)$ different hyperplanes which contain, respectively, the vertices with $1,\ 3,\ 5,\ldots,2\lceil n/2\rceil - 1$ ones among their elements.

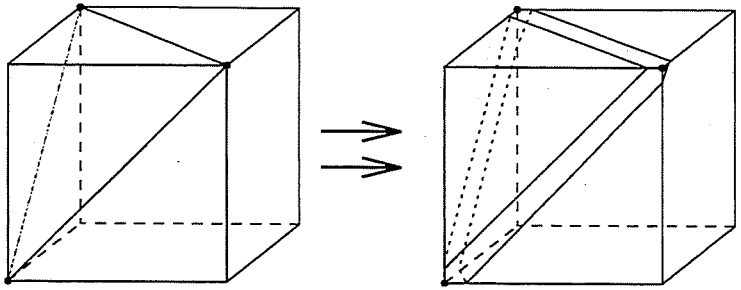
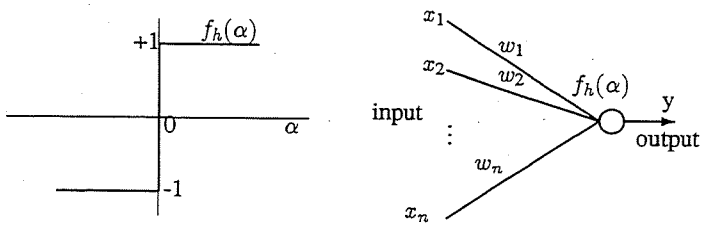


Figure 2.1: A simple example of constructing two parallel hyperplanes from one hyperplane in 3-dimensional space.

With respect to each of the above p hyperplanes, we can construct two parallel hyperplanes, each of which is obtained by shifting over a infinitesimal distance parallel to the original hyperplane in two opposite directions. Such construction can ensure that no even vertices lie in the slice between the two constructed hyperplanes, and each of the odd vertices is contained in one of these hyperplane slices. Returning to the mathematical expression of the hyperplanes in Equation (2.1), the two infinitesimally shifted hyperplanes can be obtained by setting b to be $b - \epsilon$ and $b + \epsilon$, where ϵ is a small constant. Clearly, such a construction will result in $2p$ hyperplanes. To illustrate, an example in 3-dimensional space is shown in Figure 2.1 in which the small constant ϵ represents the distance between the two parallel hyperplanes.

Recall that in a multilayer perceptron each neuron with n inputs and a single output acts like a hyperplane in n dimensional input space, and the inverse is also true[56]. For example, the hyperplane represented by Equation (2.1) can be implemented by a neuron with n inputs, with the weights from each of these n inputs to the neuron to be $a_1, a_2, \cdots, a_n$ and the threshold of the neuron to be b. The output function of the neuron is determined based on the individual application. Here we choose the function to be "hard limiter", see Figure 2.2 for a graphic definition. It follows that for the ith pair of the hyperplanes built in the way outlined above, we can always associate a pair of neurons, denoted by v_1^i and v_2^i, so that any vertices falling in between these two



HARD LIMITER $y = f_h(\sum_{i=1}^{n} w_i x_i - \theta)$

Figure 2.2: The Hard Limiter processing function for neurons and a neuron with n inputs and one output.

hyperplanes will give rise to the output $+1$ for the neuron v_1^i, and -1 for the neuron v_2^i. Moreover, since all the rest vertices (those not falling in between these two hyperplanes) always lie on the same side of the two hyperplanes (either above or under them), they can only exert an identical output for both of the neurons (either $+1$ or -1, depending on which side of the two hyperplanes they lie on). By so doing, we come up with a line of $2p$ neurons.

It remains to determine the weights and the threshold value for the output neuron, say y, to implement the final classification. This is easily done as follows. The weight from v_1^i to y is set to $+1$, and the weight from v_2^i to y is set to -1, and the threshold of the output neuron y is set at 0.5. From the above construction it can be seen that each of the odd vertices will always fall in between one of those p hyperplane slices, while no even vertices fall in between any of the hyperplane slices (they are always outside them). It then follows from the weight and threshold setting of the network that any odd vertex which always lies in between one of the hyperplane slices will give rise to the output 1 for y, and similarly, any even vertex which always lies outside any of the slices can only result in the output of -1.

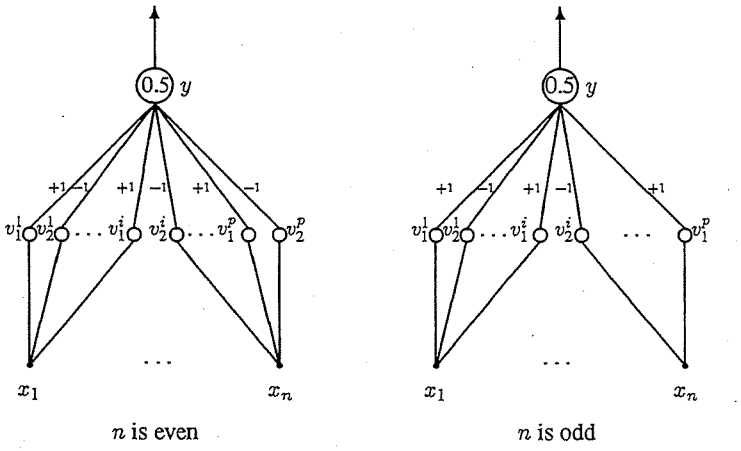


Figure 2.3: One-hidden-layer perceptron with n hidden neurons and one output neuron (with hard limiter function) thresholding at 0.5 solving the n-parity problem in an n-dimensional hypercube.

Concerning the number of neurons used, note that $2p = n$ when n is even, $2p = n + 1$ when n is odd. In the case that n is odd, there is only one vertex, say q, which has n 1's (all the elements of this vertex are 1's). This case corresponds to the following hyperplane (by setting k to be n in (2.2)):

$$x_1 + x_2 + \cdots + x_n = n. \tag{2.3}$$

Clearly, for this special hyperplane, all the other vertices except vertex q (which lies on it) lie on the same side of it. So instead of constructing a hyperplane slice with two hyperplanes in the way outlined above, we just need to shift the hyperplane (2.3) a little bit such that vertex q and all the other vertices lie on the different sides of the moved hyperplane. So it can be separated from the rest of the vertices using one hyperplane, or equivalently, one neuron. The output of this neuron will be $+1$ for q, and -1 for the rest of the vertices. Subsequently, the output neuron y receives an input sum -1 for even vertices and 1 for odd vertices which can be separated by the output neuron y thresholding at 0.5. Therefore, only $2p - 1 = n$ neurons are actually needed in this case (see Figure 2.3). Summarizing the above discussion, we arrive at the following theorem:

Theorem 1 *A one-hidden-layer perceptron with n hidden neurons can solve the n-parity dichotomy problem in an n-dimensional hypercube.*

2.3 Mapping an arbitrary dichotomy to an n-parity problem

The aim of this section is to design a one-layer perceptron to map an arbitrary dichotomy problem in m-dimensional hypercube to an n-parity problem, here n is larger than m. The design method to be presented constitutes one of the two major steps in our general design approach. The obtained one-layer network will be cascaded to the inputs of the n-parity network designed in the last section to finalize the whole network for an arbitrary dichotomy problem.

Let us recall that the Hamming distance is defined to be the number of the different bits between two binary vectors. To begin the design, we first partition the given 2^m vertices into a number of sets $S(V)$ in such a way that all the vertices in one set have the Hamming distance equal to 1 with a fixed vertex

V ($V \notin S(V)$) (this fixed vertex is called the central vertex of the set, but it doesn't belong to this set). Note that if the central vertices are suitably chosen and their number is sufficiently large, all the 2^m vertices will at least belong to one of the sets. Further, we can always adjust the partition of the sets so that each of the sets only includes m vertices. With such a partitioning, it follows that any two vertices in the same set have the Hamming distance 2 to each other. Furthermore, the inner product of any vertex in set $S(V)$ with its central vertex V is $m-2$ (since they are one Hamming distance apart), while the inner product of V with any vertex which doesn't belong to this set are different from $m-2$. From the above observations we can obtain another interesting property of this partitioning which is given in the following theorem:

Theorem 2 *All the m vertices in each set $S(V)$ lie on an $(m-1)$-dimensional hyperplane in the m-dimensional space, and moreover, the vertices which do not belong to this set will not lie on this hyperplane.*

In the following, we will give a constructive proof of the theorem and hence find such a hyperplane to satisfy the two conditions, and then finally, prove that the construction is unique.

Proof:

Suppose the hyperplane on which the m vertices of a set lie has the following form

$$a_1 y_1 + a_2 y_2 + \cdots + a_m y_m = 1, \tag{2.4}$$

where a_i ($i = 1,2,\ldots,m$) are the coefficients of the hyperplane, and y_i ($i = 1,2,\ldots,m$) are the scalar variables. We select an arbitrary set $S(V)$, with central vertex V. Then the m vertices in this set $X_1, X_2, \ldots, X_m$ should satisfy the Equation (2.4). Note that each of the X_i ($i = 1,2,\cdots,m$) is an m-dimensional column vector.

Upon defining the m-dimensional vectors $A_r = (a_1, a_2, \ldots, a_m)^T$ (coefficient vector of the hyperplane), $E = (1,1,\ldots,1)^T$, and the $m \times m$ matrix

$$X = \begin{bmatrix} X_1^T \\ X_2^T \\ \vdots \\ X_m^T \end{bmatrix},$$

and noting that X_i ($i = 1,2,\ldots,m$) should satisfy (2.4), we can write down the following set of linear equations with the coefficients of the hyperplane as unknowns:

$$XA_r = E. \tag{2.5}$$

Recall that the inner product between each X_i and V is $m-2$, or equivalently,

$$E = \frac{XV}{m-2}. \tag{2.6}$$

Substituting (2.6) into (2.5) and move everything to the left-hand side, we have:

$$X(A_r - \frac{V}{m-2}) = 0. \tag{2.7}$$

Since matrix X has full rank, the unique solution to the above equations is

$$A_r = \frac{V}{m-2}, \tag{2.8}$$

hence we constructed a hyperplane on which all the m vertices lie:

$$V^T Y = m-2. \tag{2.9}$$

On the other hand, any vertex Q which does not belong to the set $S(V)$ will be such that $V^T Q \neq m-2$ which means it does not lie on the hyperplane (2.9).

The proof is thus complete. □

Based on the hyperplane constructed above, we will now construct a new hyperplane which can separate the m vertices in set $S(V)$ into two arbitraryly specified subsets $S(V)_1$ and $S(V)_2$ such that all the vertices in $S(V)_1$ will lie above the new hyperplane and those in $S(V)_2$ will lie under it. For a particular vertex in $S(V)$, it will be grouped to either subset $S(V)_1$ or $S(V)_2$, depending on which of the two output classes this vertex belongs to and whether there are an even or odd number of hyperplanes already lie under this vertex.

Suppose that the original classification problem is to classify 2^m vertices into two classes, say, class A and class B. Recall that we will first map this problem to an n-parity problem. Further suppose that all the class A vertices

will be mapped to odd vertices in n-dimensional hypercube and all the class B vertices will be mapped to even vertices in n-dimensional. Particular to the m vertices in set $S(V)$ which are lying for the moment on the hyperplane constructed above, the purpose of the newly constructed (or rotated) hyperplane is the following:

1. The rotation should be sufficiently small so that the relative position between the vertices not in set $S(V)$ and the rotated hyperplane is not affected.

2. The rotated hyperplane should make the vertices in $S(V)_1$ lie above and the vertices in $S(V)_2$ lie under it, while the subsets $S(V)_1$ and $S(V)_2$ are obtained as follows: If a vertex is to be classified in class A (i.e., mapped to odd vertex in n-dimensional hypercube), and if it already lies above an odd number of hyperplanes, then put it into $S(V)_2$, otherwise (i.e., it already lies above an even number of hyperplanes), put it into $S(V)_1$; If a vertex is to be classified in class B (i.e., mapped to even vertex in n-dimensional hypercube), and if it already lies above an odd number of hyperplanes, then put it into $S(V)_1$, otherwise (i.e., it already lies above an even number of hyperplanes), put it into $S(V)_2$.

When this is done, we shall obtain, say, ℓ hyperplanes (ℓ will precisely be the number of sets of the partition we have introduced) each of which corresponds to a single neuron in the only layer of the perceptron (recall that each neuron acts as a hyperplane in the input space).

In the following we will discuss how exactly the rotated hyperplane is constructed. Denote the entries of V which is the central vertex of set $S(V)$ by $v_1,\ v_2, \ldots,\ v_m$. Since every vertex in set $S(V)$ has only one entry different from V (recall that their Hamming distance is 1 by the definition), the vertices in this set (V itself is not in this set) can as well be expressed as

$$\begin{aligned} X_1 &= (x_{11},\ v_2,\ v_3, \ldots, v_m)^T \\ X_2 &= (v_1,\ x_{22},\ v_3, \ldots, v_m)^T \\ &\cdots \\ X_m &= (v_1,\ v_2, \ldots, v_{m-1},\ x_{mm})^T \end{aligned}$$

where x_{ii} is the ith entry for X_i. Suppose P vertices $X_{i_1}, X_{i_2}, \ldots, X_{i_P}$ are to be classified into $S(V)_1$. To do so, we rotate in an infinitesimal angle the

hyperplane (2.9) so that the relative positions (above or under) between the rotated hyperplane and all the vertices not in the set $S(V)$ will not be affected, and moreover, the vertices in $S(V)_1$ will lie above the rotated hyperplane and the vertices in $S(V)_2$ will lie under the rotated hyperplane.

More precisely, based on the hyperplane (2.9), we will do the rotation in the following way. Let ϵ (> 0) be a sufficiently small constant number and θ be a constant. The rotated hyperplane will have the following form:

$$V'^T Y = \theta, \tag{2.10}$$

and the coefficient vector V' of the rotated hyperplane is constructed in the following way:

$$\begin{aligned} V'^T \ = \ & [v_1 + Sgn(v_1)\epsilon,\ v_2 + Sgn(v_2)\epsilon, \ldots, \\ & v_{i_1-1} + Sgn(v_{i_1-1})\epsilon,\ v_{i_1} - Sgn(v_{i_1})\epsilon,\ v_{i_1+1} + Sgn(v_{i_1+1})\epsilon, \ldots, \\ & v_{i_2-1} + Sgn(v_{i_2-1})\epsilon,\ v_{i_2} - Sgn(v_{i_2})\epsilon,\ v_{i_2+1} + Sgn(v_{i_2+1})\epsilon, \ldots, , \\ & v_{i_P-1} + Sgn(v_{i_P-1})\epsilon,\ v_{i_P} - Sgn(v_{i_P})\epsilon,\ v_{i_P+1} + Sgn(v_{i_P+1})\epsilon, \\ & \ldots, v_m + Sgn(v_m)\epsilon] \end{aligned} \tag{2.11}$$

where $Sgn(\cdot)$ is the sign function.

Substituting each X_i $(i = i_1, i_2, \ldots, i_P)$ (vertices to be classified into the subset $S(V)_1$) into (2.10) results in

$$V'^T X_i = m - 2 + [m - (P-1)]\epsilon - (P-1)\epsilon = m - 2 + [m - 2(P-1)]\epsilon.$$

Substituting each X_i $(i \neq i_1, i_2, \ldots, i_P)$ (vertices to be classified into the subset $S(V)_2$) into (2.10) results in

$$V'^T X_i = m - 2 + [m - (P+1)]\epsilon - (P+1)\epsilon = m - 2 + [m - 2(P+1)]\epsilon.$$

Clearly θ can be set to any values between $m - 2 + [m - 2(P-1)]\epsilon$ and $m - 2 + [m - 2(P+1)]\epsilon$ to finish the construction of the rotated hyperplane. Here we choose θ to be the average of the two values which is

$$\theta = m - 2 + (m - 2P)\epsilon. \tag{2.12}$$

It is not difficult to verify that with such a setting for V' and θ in (2.11) and (2.12), the vertices in subset $S(V)_1$ will lie above the hyperplane (2.10) and the vertices in subset $S(V)_2$ will lie under the hyperplane (2.10).

If we construct this type of hyperplanes for all the sets, then every vertex in the input m-dimensional hypercube can be located above the even or odd number of the hyperplanes, thus can be mapped into the even or odd vertex of an n-dimensional hypercube using a single layer perceptron network.

Connecting the outputs of this single layer perceptron in this section with the inputs of the one-hidden-layer perceptron constructed in the previous section, we can obtain a two-hidden-layer perceptron networks with m inputs and 1 output. This two-hidden-layer structure can implement an arbitrary dichotomy among the 2^m binary inputs.

2.4 An example

In this section we will use the design procedure discussed above to actually design a two-hidden-layer network for a binary dichotomy problem in a 3-dimensional space. The task is to classify all the vertices in the 3-dimensional hypercube into two classes, i.e.,

$$\textbf{Class 1:} \quad \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}$$

$$\textbf{Class 2:} \quad \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Step 1 The first step in the procedure is to group all the 2^m (here $m = 3$) vertices into a number of sets $S(V_i)$, where V_i is the central vertex. It's easy to see that by choosing the following central vertices $V_1 = [-1,-1,-1]'$, $V_2 = [-1,-1,1]'$, $V_3 = [-1,1-1]'$ and $V_4 = [-1,1,1]'$, all the 8 vertices are at least in one of the four sets. The detailed partitioning of the vertices are in the following:

$$V_1 = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}, \quad S(V_1) = \left\{ \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \right\}$$

$$V_2 = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}, \quad S(V_2) = \left\{ \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} \right\}$$

$$V_3 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, \quad S(V_3) = \left\{ \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} \right\}$$

$$V_4 = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}, \quad S(V_4) = \left\{ \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right\}$$

Step 2 Using Equation (2.9) to find the four hyperplanes l_1, l_2, l_3 and l_4 on which each of the four sets of vertices lie:

$$l_1: \; -y_1 - y_2 - y_3 = 1$$

$$l_2: \; -y_1 - y_2 + y_3 = 1$$

$$l_3: \; -y_1 + y_2 - y_3 = 1$$

$$l_4: \; -y_1 + y_2 + y_3 = 1$$

Step 3 To map this dichotomy problem first to an n-parity (here $n = 4$) problem, we need to place an even number of planes under the vertices in **Class 1** and an odd number of planes under the vertices in **Class 2** by slightly rotating[1] the four planes, hence we have the following table:

[1] Sometimes when all the vertices in a set need to be placed on the same side of the hyperplane, shifting, which is much simpler than rotating, can be used instead.

	before rotation				after rotation			
vertex	l_1	l_2	l_3	l_4	l_1	l_2	l_3	l_4
$[-1\ -1\ -1]'$	>	○	○	<	>	>	<	<
$[-1\ -1\ \ 1]'$	○	>	<	○	>	>	<	<
$[-1\ \ 1\ -1]'$	○	<	>	○	>	<	>	<
$[-1\ \ 1\ \ 1]'$	<	○	○	>	<	>	<	>
$[\ 1\ -1\ -1]'$	○	<	<	<	>	<	<	<
$[\ 1\ -1\ \ 1]'$	<	○	<	<	<	>	<	<
$[\ 1\ \ 1\ -1]'$	<	<	○	<	<	<	>	<
$[\ 1\ \ 1\ -1]'$	<	<	<	○	<	<	<	>

Note: in the above table ">" means the vertex is above the corresponding plane, "<" means the vertex is under the corresponding plane and "○" means the vertex lies on the corresponding plane. "before rotation" means the relative positions between the vertices and the planes before the rotation and "after rotation" means the required relative positions between the vertices and the planes once the rotations are done.

From the right half of the above table it is easy to see the relative positions between the vertices and the plane of each of the four sets which are the following:

- For $S(V_1)$, all the vertices should be above l_1.
- For $S(V_2)$, all the vertices should be above l_2.
- For $S(V_3)$, $[1\ 1\ -1]'$ should be above, $[-1\ -1\ -1]'$ and $[-1\ 1\ 1]'$ should be under l_3.
- For $S(V_4)$, $[1\ 1\ 1]'$ should be above, $[-1\ -1\ 1]'$ and $[-1\ 1\ -1]'$ should be under l_4.

Step 4 Since all the vertices in $S(V_1)$, respectively $S(V_2)$, should be above the moved plane l_1', respectively l_2', a simple shift, instead of rotation, will do the job. So we have

$$l_1': \quad -y_1 - y_2 - y_3 = 0.9 \tag{2.13}$$

$$l_2': \quad -y_1 - y_2 + y_3 = 0.9 \tag{2.14}$$

Using Equation (2.11) (taking ϵ to be 0.1), we can construct the rotated planes l_3' and l_4' from l_3 and l_4 as following:

$$l_3': \quad -0.9y_1 + 1.1y_2 - 1.1y_3 = 1.1 \tag{2.15}$$

$$l_4': \quad -0.9y_1 + 1.1y_2 + 1.1y_3 = 1.1 \tag{2.16}$$

The parameters of the planes l_1', l_2', l_3' and l_4' will be the weights and the thresholds for the first hidden layer of the network and the original dichotomy problem becomes a 4-parity problem.

Step 5 From the right half of the above table, it can be seen that the vertices in **Class 1** (top 4) have outputs with even number of +1's for the first hidden layer, while the vertices in **Class 2** (bottom 4) have outputs with odd number of +1's (here they all have one +1 in the outputs) for the first hidden layer.

Using Equation (2.2) we can construct a hyperplane on which all the four first hidden layer outputs of the vertices in **Class 2** will lie as following:

$$x_1 + x_2 + x_3 + x_4 = -2. \tag{2.17}$$

Two parallel hyperplanes can then be constructed based on hyperplane (2.17) as following:

$$x_1 + x_2 + x_3 + x_4 = -1.9 \tag{2.18}$$

$$x_1 + x_2 + x_3 + x_4 = -2.1. \tag{2.19}$$

Since all the odd vertices (outputs of the first hidden layer) have the same number of +1's (here it is 1), one pair of such hyperplanes (2.18) and (2.19) are enough for solving the 4-parity problems, which means two neurons are needed in the second hidden layer of the network. The parameters in (2.18) and (2.19) will be the weights and thresholds for the second hidden layer neurons.

Step 6 Finally, using the structure in Figure 2.3 (n is even), we can construct a complete network with four neurons in the first hidden layer, two neurons in the second hidden layer and one output neuron as shown in Figure 2.4. Note that all the neurons in the figure have the output function as shown in Figure 2.2.

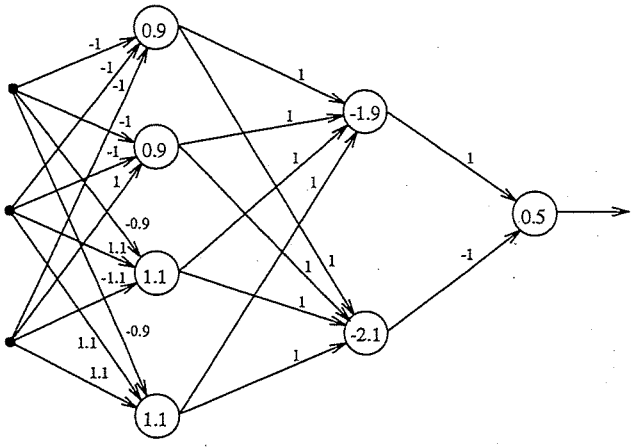


Figure 2.4: The neural net for a dichotomy problem in 3-dimensional hypercube.

2.5 Discussions and conclusions

There are two major approaches to design a multilayer perceptron for practical tasks. One is to apply some kind of learning algorithm, the other is to use direct setting strategies which are inspired by analytic and geometric considerations. One of the drawbacks suffered by learning algorithms is that the determination of the network structure is based on trail-and-error, and there are no general guidelines available. This often results in the over or under determination of the structures. Another drawback is that it generally takes a long time for learning algorithms to converge, or even more seriously, some learning algorithms are intrinsically not able to find the weights for certain dichotomy problems (see, for example [7]).

The direct approach can avoid the aforementioned problems, but will generally require the detailed understanding of the nature of both the problem and the net structure. The geometrical approach we have proposed belongs to this second category. Obviously, the detailed geometrical analysis has enabled us to come up with a quite systematic method for the determination of net

structures and the setting-up of weights and thresholds. It should be noted that our net structure is aimed at arbitrary problems, and a much smaller net might be obtained by exploring the nature of some specific problems. Obviously, the reduction is achieved at the expense of losing the generality. In this sense, our network structure provides a meaningful upper-bound (or more precisely, the maximum necessary) to the multilayer perceptron used in this purpose.

It would be interesting to know the total number of neurons used in our construction. Unfortunately, this is still an open problem as it depends on the construction of the number of sets introduced in Section 2.3. Up till now, we are yet to find the ways to establish the minimal possible number of such sets. Nonetheless, it is known that this number greatly depends on the way the central vertices are selected and is always greater than $2^m/m$ (m is the dimension of the input hypercube).

Chapter 3

Recursive Network Design: Associative Memories

In this chapter, we will discuss the design of recursive neural networks as associative memories. After giving a short introduction to the concept of associative memories and reviewing the famous model of Hopfield associative memories, we make our contributions to this topic mainly in the following aspects:

1. We developed a systematic method for the fixed-point programming, which is the basic and most important requirement for an associative memory system. The method we used is to first translate the problem of fixed-point programming to solving a set of inequalities and then use multilayer feedforward networks and the backpropagation algorithm to solve these inequalities.

2. We introduced the concept of hidden structures to the standard Hopfield associative memories and hence expanded the storage ability of the original model. We also gave a constructive proof via which an arbitrarily given set of binary vectors can be made orthogonal to each other and storable in an expanded Hopfield associative memories.

3. We proposed an optimization based weight updating rule for Hopfield associative memories. This method not only provides an explanation

about the relations among the different parameters in a Hopfield associative memory (e.g., weights, fixed-points, sizes of attracting basins, activations, etc.), it also maximizes the sizes of certain type of attracting basins.

4. We studied the possibilities of using cellular neural networks as a model of associative memories and gave a most straight forward template for a simple cellular neural network model of associative memories.

3.1 Introduction

In this section, we will first give a short introduction to associative memories, and then we will give an overview of Hopfield associative memories, about its structure and design rule. We will also review the existing results on this topic and introduce two formulations of the problems concerning Hopfield associative memories. In the last we will summarize the results obtained in this chapter and outline the remaining sections.

3.1.1 Associative memory

For many years researchers in various disciplines have been studying models for the brain. Many models have been developed in an attempt to understand how human neural networks function. One class of such models is based on the concept of *associative memory* (AM). Associative memories are composed of a collection of interconnected elements having data storage capabilities, and they are designed for the storage and retrieval of items of information, which are called *patterns*. Each pattern to be stored must be identified with another pattern (probably in different form), to be used as *key* or address to retrieve the first form of the pattern stored previously. If the second pattern has the same form with the first pattern but is just a portion of it (e.g., for a pattern in vector form, the first pattern is an n-dimensional vector, then the second pattern will be an m-dimensional ($m < n$) vector with its elements being the counterparts of the first pattern), this associative memory is called a *content-addressable memory* (CAM) or *autoassociative memory*.

For the purpose of comparison, in a digital computer a desired piece of information, called a *memory*, is recalled when the correct address of the

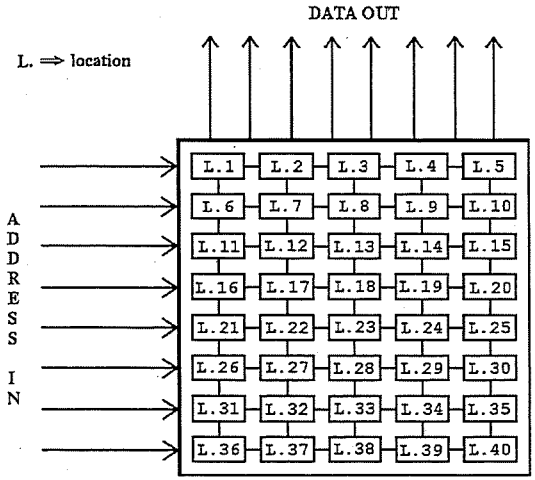


Figure 3.1: An address addressable memory with each location storing one pattern. With the eight address lines shown in the figure, there could be up to 256 locations to store patterns, while only 40 of them are shown here.

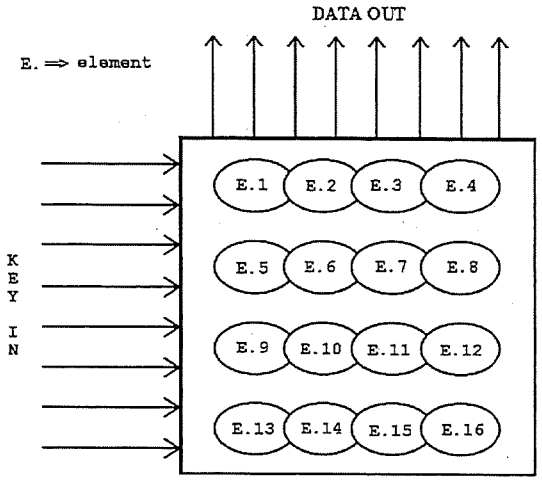


Figure 3.2: An associative memory with each element storing a portion of multi patterns and each pattern is stored over multi elements.

memory is given. This is called *address addressable memory* (AAM). In contrast, an associative memory is able to recall a complete pattern of the memory when the system (e.g., a neural network associative memory) is excited by a sufficiently large portion of that pattern, called a *key*. Thus, a complete pattern of the memory is recalled by a portion of the pattern. For example, if each text line in the "Contents Table" of this thesis is stored in an associative memory, and the key "of Hopfield nets by solving" is given, then the associative memory will respond with the complete set of information which is "3.2.3 Design of Hopfield nets by solving inequalities". The difference between an address addressable memory (AAM) and an associative memory (AM) has been illustrated in Figure 3.1 and Figure 3.2.

The subject of this chapter will mainly be autoassociative memories, sometimes simply referred to as associative memories. According to Kohonen ([51]), an ideal autoassociative memory is defined as follows: It is a system which holds copies of distinct input signal sets $x^{(p)}, p = 1, 2, \cdots, k$ in its internal state, and produces the copy of a particular set $x^{(r)} = (\zeta_1^{(r)}, \zeta_2^{(r)}, \cdots, \zeta_n^{(r)})$, $r \in \{1, 2, \cdots, k\}$ to the outputs, whenever (in the recall mode) the inputs are excited by a set of signals $x = (\zeta_1, \zeta_2, \cdots, \zeta_n)$ in which a specified subset of the values ζ_i matches with the corresponding subset of the $\zeta_i^{(r)}$.

An efficient AM can store a large set of memory patterns and recall each memory pattern when the system is excited by a key containing a portion of that memory's information. For comparison, a completely correct address is required to recall information from an AAM. Moreover, if a sufficient amount of correct memory information is provided, an AM may be able to correct the erroneous data. On the other hand, an AAM will always provide the correct information if the entire correct address is given, whereas in an AM there is no guarantee that the output will always constitute correct information.

One key factor of associative memories is the way the information is stored: Each element of the memory contributes to the storage of more than one pattern as shown in Figure 3.2 (sometimes referred to as *distributed* storage). The logical structure of an AAM, such as those in a conventional digital computer, is straightforward (see Figure 3.1). Each piece of information is stored in a separate location and can be retrieved with perfect accuracy, and the quantity of information that can be stored is simply determined by the number of available locations (can be referred to as *local* storage). The design of a *distributed* memory presents problems. By definition each memory element can be part of several patterns, and each pattern is distributed over more than one memory elements. Thus the retrieval process involves more than just reading out from a sequence of memory locations. The key problem here is how the information should be stored so that it can be retrieved accurately.

Many models of associative memories have been proposed and studied and many methods have been developed to design distributed memory systems ([51][38][47]). Characterized by parallel computation and dynamical convergence, recurrent neural networks have been studied to function as a model of associative memories. In 1982 and 1984, Hopfield ([42][43]) introduced a new description of associative or content-addressable memory, based on his

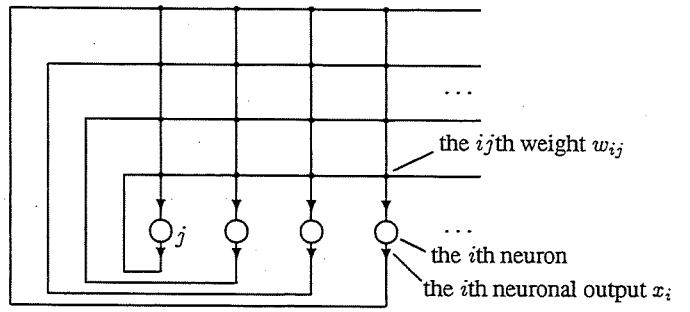


Figure 3.3: The configuration of the Hopfield net. The thresholding function for each neuron is assumed to be the Sign function.

studies of collective computation in neural networks. After that, Kosko ([52]) proposed his structure of *Bidirectional Associative Memories* for paired-data associations in 1988. Since then, much work has been done on models of the neural associative memories, especially on the Hopfield model and potential applications in areas such as, image processing, data base, text retrieval and complex minimization problems, etc. (see [4] for an example) are studied.

3.1.2 The Hopfield model of associative memories

A. **The Hopfield model**

Storing patterns as the stable fixed points of a dynamical process and later retrieving them by means of dynamical convergence is the essential idea behind some recently developed models of neural associative memories. To be of any practical significance, models of this kind should provide an easy mechanism to affect, preferably in an arbitrary fashion, their dynamical features such as locations of the stable equilibria, sizes of attracting basins, etc.

Representative among all such models is the famous Hopfield net. The configuration of a standard Hopfield net is quite simple (see Figure 3.3). An n-neuron Hopfield net consists of n threshold elements (called neurons) that

are fully connected via weighted links. The threshold function $T(\cdot)$ for the ith neuron is simply chosen as a Sign function, i.e., $\mathrm{T}(u-\theta_i) = -1, u < \theta_i$ and $\mathrm{T}(u-\theta_i) = 1, u \geq \theta_i$, where u is the input and θ_i is the threshold of the neuron. All the connecting weights can be arranged into an $n \times n$ matrix W with the entry w_{ij} denoting the weight from the ith neuron to the jth one. Similarly, all the neuronal outputs, i.e., the outputs of the linear threshold elements in the net, can be arranged into an n-dimensional vector x with x_i being the ith neuronal output (unless otherwise specified, vectors in this thesis are assumed to be column vectors). W and x will be called, respectively, the *weight matrix* and the *state vector* of the net. The vector θ with ith entry being θ_i will be called the *threshold vector* of the net.

Because of the thresholding, each neuronal output can only assume two different output values, -1 and 1 in our case. Thus x is always a binary bipolar vector. The neuronal states are updated either synchronously or asynchronously. Given the state vector x^t at time t of a Hopfield net, the input vector of the net will be $W \cdot x^t$. When a synchronous updating rule is adopted, the new state vector of the net will be $x^{t+1} = f(W \cdot x^t - \theta)$. While under the asynchronous updating rule, only one single neuron is randomly picked up and its state is updated at each discrete time instant. One of the most important properties of Hopfield nets is that there exist a number of stable equilibria with certain size of attracting basins which can be used to store binary patterns.

B. The Hebbian rule

As presented in the above, the Hopfield net turns out to define a discrete dynamical process when the weight matrix W, the threshold vector θ and the updating rule are fixed. Clearly, the precise nature of this dynamical process is affected by the connecting weights and the thresholds of the neurons.

Given a set of binary n-dimensional column vectors $f^1, f^2, \ldots, f^k$, the conventional design problem of the Hopfield net is formulated as finding a weight matrix W and a threshold vector θ so that the given vectors are made the stable fixed points of the net. Note that the issue of the sizes of attracting basins around these stable fixed points is not explicitly concerned in this formulation. One of the problems associated with Hopfield associative memories is that while the given binary patterns are made the stable fixed points of the net, there are still some extra unwanted stable fixed points and these are

called *spurious states*. The stability of the network is always required for the application of associative memories. Whenever the network is activated by an whatever input pattern, the network should always converge to a stable fixed point, no other operation modes (like *limit-cycle or chaos*) are allowed. This is defined as the global network stability. The issues of global stability and reducing the number of spurious states are tackled in the above formulation by superimposing certain structure on W, e.g., symmetry and zero diagonal elements.

Blind search for an appropriate W and θ can be extremely hard and ineffective. The design, therefore, proceeds by postulating certain weight set-up rules and then try to see if they lead to the desired consequence. One noticeable example of this sort of design rules is the Hebbian rule (see [42][59]). For a given set of binary patterns $f^1, f^2, \ldots, f^k$ of dimension n, the Hebbian rule sets the threshold vector θ to 0 and constructs the weight matrix in the following way

$$W = \sum_{i=1}^{k} (f^i (f^i)^T - I_n), \tag{3.1}$$

where I_n is the identity matrix of dimension n. (3.1) is essentially the sum-of-outer-product of the given set of patterns, except that the diagonal entries are set to zero (a heuristic justification for this is that it will reduce the number of spurious states). Obviously, the weight matrix obtained this way is symmetric, which together with the zero-diagonal, guarantees the global stability of the net when the net undergoes asynchronous state update. Other rules of the same nature, but of more recent origin can be found in [17][102].

A modified version of the Hebbian rule called *clipped Hebbian rule* is introduced in [18]. This simplified design rule can be described as follow: If W is the weight matrix obtained using the Hebbian rule, then

$$W' = G(W) \tag{3.2}$$

will be the weight matrix obtained using the *clipped Hebbian rule*, where $G(\cdot)$ is a function defined as $G(x) = -1, x < 0$; $G(x) = 1, x > 0$; $G(x) = 0, x = 0$. The properties and performances of the *clipped Hebbian rule* are discussed in [18] and the references therein. The general observation is that this special design rule is basically the same as the original Hebbian rule, but the fact that

each weight can only assume three different values does give advantages over the original Hebbian rule to the VLSI implementation of the weights.

There exists a simpler way to obtain W' without resorting directly to the sum-of-outer-product. Each row j of W' can be obtained by adding together all the given patterns with the jth entry being $+1$ and subtracting this sum by all the given patterns with the jth entry being -1, setting the jth entry of the jth row of W' to 0, and then taking the $G(\cdot)$ function on the result. Note that all the adding and subtracting operations here are vector operations. This method of computing W' can also be expressed mathematically in the following:

Let f_j^i $(i = 1,2,\cdots,k,\ j = 1,2,\cdots,n)$ be the jth element of ith given pattern. Define an intermediate matrix U with its jth $(j = 1,2,\cdots,n)$ row U_j being

$$U_j = \sum_{i=1}^{k} [\text{Sign}(f_j^i)(f^i)^T].$$

Define another intermediate matrix V with the elements of its jth $(j = 1,2,\cdots,n)$ row V_j being those of U_j except that the jth element of V_j is set to zero. Then

$$W' = G(V), \tag{3.3}$$

where $G(\cdot)$ is the same function as we defined previously.

Although the performance of the Hebbian rule is not so impressive (this point will be detailed in what follows) and far from practically usable, it has been the most widely used design rule for Hopfield associative memories due to the lack of other efficient rules. This problem has been the main obstacle in the development of this area for many years.

C. Remarks on the standard Hopfield model

As presented previously, a standard Hopfield associative memory is designed using the Hebbian rule or its modified version. While the Hebbian rule may produce a stable net with less spurious states, there is no guarantee that the given binary patterns f^i's $(i = 1,2,\cdots,k)$ be made the fixed points of the net, let alone the problem of controlling the sizes of attracting basins.

The maximal number of an arbitraryly given set of binary patterns that can be made the stable fixed points of a Hopfield network is usually called

the *capacity* of the network [65]. It has been shown that even without any requirement on the sizes of attracting basins, the capacity of the Hopfield net designed using the Hebbian rule will vary wildly depending on the specific nature of the given set of patterns (see [17][10] for simple illustrations). In [10] it has been demonstrated that as few as two patterns can not be simultaneously made the fixed points of a Hopfield net by the Hebbian rule, or any rule requiring the diagonal entries of weight matrix W to be zero, when the two patterns are only Hamming distance 1 apart. This case may be considered trivial as it does not allow any meaningful size of attracting basins for the patterns. A less trivial case identified in [17] shows that there even exist a few well-separated (in the Hamming distance sense) patterns that can not be simultaneously made the fixed-points of a Hopfield net.

These degenerate cases typically result when the given patterns are deterministic in nature. It thus motivated a probabilistic approach to the design of Hopfield associative memories. Initiated in [59], the probabilistic approach currently pursued in the literature ([17][102]) is founded on the following bases:

- Given patterns to be stored in a Hopfield net are assumed to be random, often a result of zero-mean Bernoulli trials (see the assumptions made in [42][59][17]). Moreover, the requirement on the attracting basin is phrased as a probabilistic condition;
- the analysis is probabilistic in nature, hence the derived results are only meaningful *asymptotically*, in the sense that the number of the given patterns may go to infinity;
- in view of the above, the dimension of the given sets of random patterns is implicitly assumed to be flexible, and can even be stretched to infinity.

When these assumptions are accepted, the storage features of the Hopfield net designed with the Hebbian rule can be rigorously derived ([59][17][102]).

Unfortunately, these assumptions do not seem to be relevant when we actually try to design an associative memory of practical interest. The basic consideration is that the given sets of patterns to be stored in the net, arbitrary though, often result from certain coding schemes, and bear some specific deterministic nature. Thus they can hardly be considered as random. Moreover,

these pattern sets necessarily have a finite and definite dimension rendering the derived asymptotic results inapplicable. It, therefore, seems that a deterministic design approach which includes explicit requirement on the fixed points and/or basins of attraction is of better practical relevance.

D. **Two formulations of the problems**

Before we discuss the two formulations of the design problems of Hopfield associative memories, we will give a definition of the size attracting basins of Hopfield associative memories in the Hamming distance sense.

Definition 2 *The size of attracting basin in Hamming distance sense is defined to be the maximal Hamming distance between a stable fixed point and all the patterns that have the same Hamming distance apart from it, and at the same time, that will converge to this stable fixed points in finite steps under a certain state updating rule.*

To design a Hopfield associative memory in a deterministic sense, the problems can be formulated into two different categories:

1. Given a set of n-dimensional binary vectors $f^1, f^2, \ldots, f^k$, and a constant integer r, find the number of linear threshold neurons m $(m \geq n)$ and an $m \times m$ weight matrix W, which can be assigned some particular structure, so that $f^1, f^2, \ldots, f^k$ (after being appendixed with certain appropriate elements to the dimension m) are stable fixed-points of the net with r being the size of the attracting basins; A similar formulation but with a weak performance requirement is: Given a set of n-dimensional binary vectors $f^1, f^2, \ldots, f^k$, find the number of linear threshold neurons m $(m \geq n)$ and an $m \times m$ symmetric zero-diagonal weight matrix W, so that $f^1, f^2, \ldots, f^k$ are stable fixed-points of the net.

2. An alternative approach is to fix, at the onset, the number of neurons in the Hopfield net being designed to be the same as the dimension of the given patterns, and to explore the connection between the structure of the weight matrix and the size of the basins of attraction (see [49] for an example). A formulation of this kind can be stated as follows: Given a set of n-dimensional binary vectors $f^1, f^2, \ldots, f^k$, and assuming that the number of the neurons to be used in the network is equal to n, find an $n \times n$ weight matrix W, which may be subject to some constraints for reasons of performance or implementation, so that $f^1, f^2, \ldots, f^k$ are stable fixed-points of the net with as large as possible attracting basins.

In fact, these two deterministic formulations of the design problems correspond to two different design principles. The first one may be called *performance-oriented*, i.e., to meet sharply all the given requirements (fixed points and/or fixed size of attraction basins) of the memory, at the expense of the size of the net; the second may be called *structure-oriented*, i.e., to constrain the size of the net while trying to obtain the best possible performance.

3.1.3 Outline of this chapter

This chapter consists of six sections. In Section 3.2, we present a systematic approach to fixed points programming by reformulating the design problem into solving a set of inequalities with the consideration of consistency. In Section 3.3, we develop a so called *data-driven* deterministic approach to the design of neural associative memories and end up with an extended Hopfield net with hidden structure. In Section 3.4, as an alternative to the Hebbian rule, a new design rule for the standard Hopfield associative memory is proposed and the results are compared to those designed using the Hebbian rule. Section 3.5 includes a new model for neural associative memory, a cellular neural network. The design rule and the issues such as capacity and size of attraction basin are discussed. In the last section, the conclusions on neural associative memories are given and additional issues are discussed.

3.2 Fixed points programming by error back propagation

In this section, we discuss the problem of making the given binary patterns to be the stable fixed points of a Hopfield net, which is the minimal requirement for an associative memory. The content in this section is structured as follows. To fix the ideas, in the subsequent subsection we briefly describe the Hopfield net and the structure of the feed-forward neural networks as we shall use in the development. In Subsection 3.2.3, the problem of finding the weight matrix

W, when k given patterns need to be stored, is reformulated in terms of a set of inequalities. The detailed procedure of using the backpropagation algorithm to design Hopfield nets together with some examples is presented in Subsection 3.2.4.

3.2.1 Introduction

The problem of fixed point programming in a discrete Hopfield associative memory can be stated as follows: Given a set of n-dimensional binary vectors $f^1, f^2, \ldots, f^k$, find the number of linear threshold neurons and the weight weight matrix of a Hopfield net so that $\{f^i\}$ $(i = 1, 2, \ldots, k)$ are made the stable fixed-points of the net. As presented in Subsection 3.1.2(C), this type of problems have mostly been studied in the probabilistic sense in the literature.

Designing discrete Hopfield nets in a deterministic framework has received relatively less attention. In fact only a limited number of treatments are available in the literature [101]. The reason is, perhaps, that the deterministic approach often requires a precise understanding of the underlying dynamics of the Hopfield nets. Such an understanding is, however, far from being achieved.

A simple observation is that the design of Hopfield nets in a deterministic framework can be formulated into solving a set of inequalities. Such a formulation is possible as long as the neurons have a linear threshold type of transfer function (see [101]). Thus the design problem reduces to finding an efficient algorithm to solve a set of inequalities. However, the derived set of inequalities may not always be consistent, raising the need to modify the basic structure of the Hopfield net.

The objective of this section is to pursue the inequality formulation of the design problem even further by considering the possibility of inconsistency. We show, via the inequality formulation, that the design of a Hopfield associative memory can be equivalently stated as designing the weights for some specially-constructed feed-forward nets. Specifically, by considering the solutions of a set of inequalities to be the target weights of a trained network, it is shown that a set of consistent inequalities can be rearranged into a two-layer feed-forward net, whereas the inconsistency will typically lead to a three-layer net with a hidden layer. We then use the well-known back-propagation algorithm to actually find the weights of the feed-forward net, thereby solving the set of inequalities (here one solution is enough).

One feature of this approach is the following: As a standard Hopfield net will not be able to memorize an arbitrary set of data, we need to modify the structure to allow hidden neurons. In fact, our design method will automatically determine the number of necessary hidden neurons as well as the weights connecting these hidden neurons.

3.2.2 Discrete Hopfield nets and feed-forward nets

As presented previously, for a Hopfield neural network to function as a desired associative memory, it is most desirable to have a design rule for the weight matrix W so that:

1. The network is stable under a certain state updating rule;
2. It makes an arbitrary set of desired memories, $f^1, f^2, \cdots, f^k$ to be stable fixed-points of the net;
3. The number of spurious states should be as small as possible;
4. Probe vectors that are close (in the Hamming distance sense) to a given pattern ought to be in its basin of attraction.

The first and third requirement in the above could be met by a symmetric zero-diagonal structure of the weight matrix W, although it is not always necessary. We shall discuss this point in the concluding section. To meet the second requirement will be the target of this section, while the fourth requirement will be discussed in the ensuing sections.

After specifying a definite state updating rule, the Hopfield net is normally turned into a recurrent net because of its feedback nature. Another important type of neural net is the so-called feed-forward net. Feed-forward neural networks are layered networks with feed-forward (unidirectionally from input to output) connections only. The specific feed-forward neural net we shall use in our approach is shown in Figure 3.4 with the nonlinearity of the neurons defined as shown.

Compared to the more familiar multilayer perceptron ([84]), the distinct feature of this feed-forward net is that there exist direct connections from the input layer $\{I_i\}$ to the output layer $\{O_i\}$, though no connection exists among I_i's or O_i's (for any i).

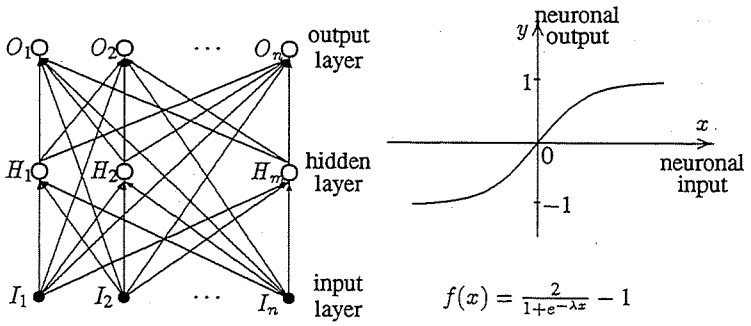


Figure 3.4: The nonlinear function of the neurons and the special feed-forward neural net structure used in the approach. Note that there are direct connections from the input nodes to the output neurons.

With the structure and the number of layers as well as the number of neurons within each layer of a feed-forward net fixed, the connection weights among the neurons are yet to be determined. The ways by which these weights are computed can roughly be divided into two categories: the direct method and the learning method. Learning method is widely used, especially when there is a great amount of training data available, and will also be used in this approach.

Given a set of inputs and the corresponding desired outputs, the well known backpropagation algorithm is often used to find the set of weights for the multilayer perceptron net ([84]). As the net shown in Figure 3.4 is not substantially different from the multilayer perceptron net, the back-propagation method can also be used to train the weights when a set of exemplar inputs and outputs are available. The details of the algorithm can be found in [56]. Moreover, like the perceptron, this feed-forward structure can be shown to be able to implement an arbitrary multi-input multi-output logic function, if the hidden layer contains enough neurons ([56][45]). We shall see that this property is closely related to the notion of the capacity in a Hopfield net.

3.2.3 Design of Hopfield nets by solving inequalities

Let k binary n-dimensional vectors $f^1, f^2, \cdots, f^k$ be given, where

$$f^i = [f_1^i, f_2^i, \cdots, f_n^i]^T, \quad f_j^i = \pm 1, \quad 1 \le i \le k, \ 1 \le j \le n.$$

Clearly, for a particular n-dimensional binary pattern f (for easy notation, we drop the superscript here) to be a fixed-point of a n-neuron Hopfield net with weight matrix W, the following should hold for each component of f.

$$\text{Sign}(\sum_{j=1}^{n} w_{ij} f_j) = \begin{cases} +1 & \text{if } f_i = +1 \\ -1 & \text{if } f_i = -1 \end{cases}, \tag{3.4}$$

or equivalently,

$$f_i \cdot (\sum_{j=1}^{n} w_{ij} f_j) > 0, \tag{3.5}$$

where $i = 1, 2, \cdots, n$. With f_i and f_j $(i, j = 1, 2, \cdots, n)$ known, (3.5) turns out to be a set of inequalities in the unknown weights w_{ij} $(i, j = 1, 2, \cdots, n)$.

For every fixed-point f^i, there exist n such inequalities. It follows that for k patterns $(f^1, f^2, \cdots, f^k)$, there are altogether $k \times n$ such inequalities. These inequalities can be written in a matrix form as follows:

$$Av > 0, \tag{3.6}$$

where

$$A = \begin{bmatrix} A_1 & & & \\ & A_2 & & \\ & & \ddots & \\ & & & A_n \end{bmatrix},$$

$$A_j = S_j \cdot F \triangleq \begin{bmatrix} f_j^1 & & & \\ & f_j^2 & & \\ & & \ddots & \\ & & & f_j^k \end{bmatrix} \cdot \begin{bmatrix} f_1^1 & f_2^1 & \cdots & f_n^1 \\ f_1^2 & f_2^2 & \cdots & f_n^2 \\ & \cdots & \cdots & \\ f_1^k & f_2^k & \cdots & f_n^k \end{bmatrix},$$

$$\begin{aligned} v &= [w_1, w_2, \cdots, w_n]^T \\ &\triangleq [0, w_{12}, \cdots, w_{1n}, |w_{21}, 0, w_{23}, \cdots, w_{2n}, | \cdots, |w_{n1}, \cdots, w_{n(n-1)}, 0]^T \end{aligned}$$

and it is required that $w_{ij} = w_{ji}$ $(i, j = 1, 2, \cdots, n)$. Note that the net symmetry and zero-diagonal are reflected in the construction of v.

Two cases need to be distinguished here:

- First, if (3.6) is a consistent set of inequalities, one can always reformulate it into a simple feed-forward neural net (without hidden layer) according to its equivalent form (3.4). As shown in Figure 3.5, the ith neuron in the output layer corresponds to the ith equation of (3.4), thus there will be n neurons in the output layer. Further, each output neuron will receive $n-1$ weighted inputs, in this case, w_{ij} becomes the unknown weight from the jth input to the ith output neuron. $w_{ii} = 0$ is reflected in the net structure by the absence of connection from the ith input to the ith output neuron, while $w_{ij} = w_{ji}$ is only guaranteed in the training process. When this correspondence is made, $f^s = [f_1^s, f_2^s, \cdots, f_n^s]^T$ naturally becomes the given input, the desired outputs corresponding to

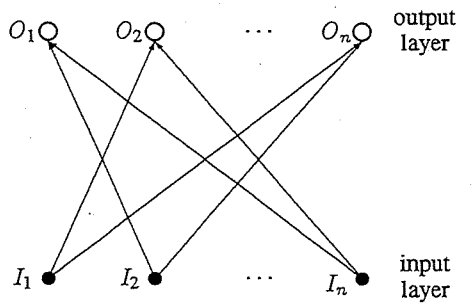


Figure 3.5: Reformulation of a consistent set of inequalities into a feed-forward net. The final weights after training this net with the back-propagation algorithm will be the solution to the inequalities.

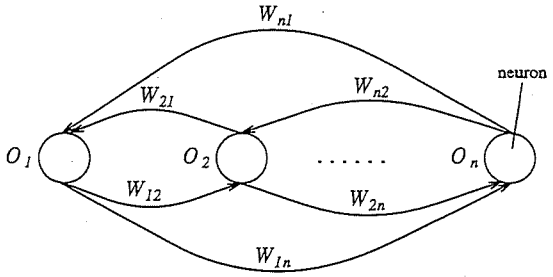


Figure 3.6: The n-neuron Hopfield net for a consistent set of data.

f^s are simply determined by the signs of its elements. Clearly, the back-propagation algorithm can then be employed to find the weights, which is, obviously, the weight matrix of the Hopfield net shown in Figure 3.6

- Secondly, if (3.6) is not consistent, which implies the inability of an n-neuron Hopfield net to store these k n-dimensional patterns, the idea of the structural redundancy detailed in the next chapter can be introduced to design a modified Hopfield net capable of storing an arbitrary set of k patterns. This modified Hopfield net is shown in Figure 3.7 with $O_1, O_2, \cdots, O_n$ being the neurons of the standard Hopfield net and $H_1, H_2, \cdots, H_m$ the hidden (redundant) neurons. There exist connections between the normal neurons and the hidden neurons, but no connections exist among the hidden neurons. It can be proved that with enough hidden neurons and appropriate selection of the redundant neuronal dynamics, an arbitrary set of k n-dimensional patterns can be coded as the stable fixed-points of the modified Hopfield net with $n+m$ neurons, where m is the number of hidden neurons used. The exact value of m depends on the design rules used. This provides the possibilities for solving the problem of the inconsistency of (3.6).

Adding hidden neurons to the standard Hopfield structure can be interpreted as introducing more variables in the inconsistent set of inequalities (3.6). But the way by which the variables are added is not arbitrary, and is actually determined by the specific neural net structure of Figure 3.7. It is easily verified that the relevant set of inequalities corresponding to the net in Figure 3.7 can be written down as

$$\bar{A}\bar{v} > 0, \tag{3.7}$$

where

$$\bar{A} = \begin{bmatrix} \bar{A}_1 & & & \\ & \bar{A}_2 & & \\ & & \ddots & \\ & & & \bar{A}_n \end{bmatrix},$$

$$\bar{A}_j = S_j \cdot [F \ \text{Sign}(F_{(k\times n)} M_{(n\times m)})],$$

$$\bar{v} = [0, w_{12} \cdots, w_{1n}, |u_{11}, u_{12}, \cdots, u_{1m}, |w_{21}, 0, w_{23}, \cdots, w_{2n}, |u_{21}, u_{22}, \cdots, u_{2m}| \cdots, |w_{n1}, w_{n2}, \cdots, w_{nn-1}, 0, |u_{n1}, u_{n2}, \cdots, u_{nm}]^T,$$

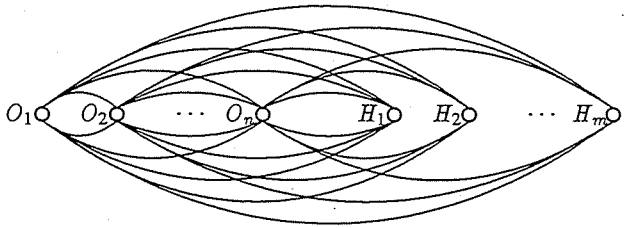


Figure 3.7: The Hopfield net for an inconsistent set of data. $O_1, O_2, \cdots, O_n$ are standard neurons while $H_1, H_2, \cdots, H_m$ are hidden neurons. Same as in Figure 3.6, these neurons are connected via weighted links, while in this case, the symbols for the weights and the directions of connections are dropped for the clarity of the figure.

in which Sign$(\cdot)$ function applies to all the entries of the input matrix, u_{ij} $(1 \le i \le n,\ 1 \le j \le m)$ is the connecting weight from the jth hidden neuron to the ith standard neuron and the element of weight matrix M, m_{ij} $(1 \le i \le n,\ 1 \le j \le m)$, is the weight from the ith standard neuron to the jth hidden neuron. In other words, for an arbitrary inconsistent set of inequalities (3.6), we can always find a number m and a matrix $M_{n\times m}$ such that (3.7) is solvable.

To construct a feed-forward net to solve (3.7), we shall first convert it to an equivalent expression with the form of (3.4) as follows

$$\text{Sign}(\sum_{j=1}^{n} w_{ij} f_j + \sum_{j=1}^{m} u_{ij} \ \text{Sign}(\sum_{i=1}^{n} m_{ij} f_i)) = \begin{cases} +1 & \text{if } f_i = +1 \\ -1 & \text{if } f_i = -1 \end{cases} \tag{3.8}$$

where $i = 1, 2, \cdots, n$. Equations (3.8) leads to a one-hidden-layer feed-forward net with n input nodes, n output and m hidden neurons as shown in Figure 3.4. The ith output neuron corresponds to the ith equation of (3.8) and Sign$(\sum_{i=1}^{n} m_{ij} f_i)$ is implemented by the jth hidden neuron. It is easy to see that w_{ij} is the weight from the jth input node to the ith output neuron, u_{ij} is

the weight from the jth hidden neuron to the ith output neuron and m_{ij} is the weight from the ith input node to the jth hidden neuron. It follows that using the same training data as in the first case and the back-propagation algorithm, the connecting weight matrices W, U and M can be obtained, which in turn determine the structure and the weights of the Hopfield net with a hidden structure.

3.2.4 Design examples

To illustrate the design method discussed above, let us consider the following two simple examples. Two sets of data are given as shown, and the task is to design appropriate Hopfield associative memories which store the given data as fixed-points.

Example 1. Consider the following set of patterns (dimension 5)

$$[1\ 1\ 1\ 1\ 1]\ [1\ 1\ 1\ -1\ -1]\ [1\ 1\ -1\ 1\ -1]\ [1\ -1\ 1\ 1\ -1] \quad (3.9)$$

First write down the set of inequalities of the form (3.6), and it can be verified that this set of inequalities is consistent. Then use the back-propagation algorithm to train a 5-neuron feed-forward net with the structure as shown in Figure 3.5. After the feed-forward net converges, the following weight matrix is obtained.

$$W = \begin{bmatrix} 0.000000 & 1.377758 & 1.303836 & 1.642787 & -2.885208 \\ 1.377758 & 0.000000 & -0.762178 & -1.160799 & 0.612523 \\ 1.303836 & -0.762178 & 0.000000 & -0.798957 & 1.186027 \\ 1.642787 & -1.160799 & -0.798957 & 0.000000 & 1.188011 \\ -2.885208 & 0.612523 & 1.186027 & 1.188011 & 0.000000 \end{bmatrix}$$

Based on W, a five-neuron Hopfield net can be set up and it can be easily checked that all the prescribed patterns (3.9) are the fixed-points of the net. A surprising additional feature is that no other stable states exist apart from the negatives of the given data. This has been checked out via exciting the designed Hopfield net by all the other patterns except the given ones, and it turns out that the Hopfield net will always converge to one of the five stable fixed points or their negatives. Unfortunately, we don't know for the moment how general this result is and can not give an explanation to this somewhat unexpected result.

Example 2. Consider the following set of patterns (dimension 5)

$$[1\ 1\ 1\ 1\ 1]\ [1\ 1\ 1\ -1\ -1]\ [1\ 1\ -1\ 1\ -1]\ [1\ -1\ -1\ -1\ 1] \quad (3.10)$$

Note that this set of data will lead to an inconsistent set of inequalities which is revealed by the non-convergence of the back-propagation trial. This indicates that a 5-neuron Hopfield net is unable to store the patterns of (3.10), no matter what design rule is used. Incorporating a one-hidden-neuron net to come up with the configuration of Figure 3.4, we can then apply the back-propagation algorithm to find the weight matrices.

$$W = \begin{bmatrix} 0.000000 & -0.199381 & -2.050556 & -0.306061 & 0.448091 \\ -0.199381 & 0.000000 & 2.016516 & 2.236412 & -3.165343 \\ -2.050556 & 2.016516 & 0.000000 & -0.933048 & 1.014154 \\ -0.306061 & 2.236412 & -0.933048 & 0.000000 & 1.093239 \\ 0.448091 & -3.165343 & 1.014154 & 1.093239 & 0.000000 \end{bmatrix}$$

$$M = \begin{bmatrix} 3.990878 \\ 1.179294 \\ 0.292349 \\ -0.680854 \\ 0.942831 \end{bmatrix}$$

$$U = \begin{bmatrix} 3.990878 & 1.179294 & 0.292349 & -0.680854 & 0.942831 \end{bmatrix}$$

These weight matrices completely define the Hopfield net. It can also be verified that a modified 6-neuron Hopfield net with the weight matrix

$$\tilde{W} = \begin{bmatrix} W & M \\ U & 0 \end{bmatrix}$$

has all the data in (3.10) as the fixed-points. Again, by trying out all the other patterns except the five given ones, no other spurious states exist except the negatives of the given patterns.

One of the technical aspect of this method is the convergence speed of the error back propagation algorithm. As it is well known, it will generally take a

very long time for the back propagation algorithm to find the global minimum, especially when the set of training data is large. Fortunately, this problem is not as serious as one may expect in our case, because the actual outputs here need not to be exactly the same with the desired outputs, but only to have the same signs with the desired outputs.

3.2.5 Conclusions

In this section we have presented a systematic approach to design a Hopfield associative memory. One of the advantages of this method is that it guarantees the prescribed patterns to be the fixed-points of the derived Hopfield net. With the idea of the structural redundancy, our approach provides a possible way to design Hopfield nets for storing arbitrary sets of patterns. An important feature of this approach is that we have successfully related a Hopfield net and its modified model with the well-known feed-forward structure, and thus the existing and relatively simple knowledge about the feed-forward net can find their way in designing more complex recurrent neural nets.

One of the untouched problems of designing a Hopfield net in this approach is the size of the attracting basins for the stable fixed points. It is known that each stored pattern should have a attraction basin with certain size and different attraction basins should not overlap with each other. One of the possible ways to tackle this problem is to introduce more inequalities to account for the direction of the attractions. An obvious drawback is that the trajectory of a probe vector needs to be specified at the onset which is not very reasonable for an associative memory. It is generally believed that weight adjustment may not be the best way to control the basins of attraction. As a result, any design rule which is purely based on weight adjustment will not be able to meet all the three requirements discussed before. This and related issues will be explored in detail in the following sections.

3.3 Pattern storage and Hopfield nets with hidden structure

In this section we will develop an alternative design approach for the Hopfield associative memory under the first formulation in Section 3.1.2. This approach

is founded on a data-driven consideration and is deterministic in nature. Based on the unambiguous design specifications we propose a series of modifications to the standard Hopfield model resulting in the so-called *expanded Hopfield net*. The structure of the expanded Hopfield net is simply that of the standard Hopfield net appended with a subnet known as the hidden structure. We will extend the Hebbian rule to set up the weights for this extended net by an orthogonalizing technique. We also derive the precise structural conditions under which the net can be designed to store any set of patterns with prescribed size of attracting basins.

3.3.1 Introduction

Given a set of binary vectors $p^1, p^2, \ldots, p^m$ and a prescribed retrieval range r (in the sense of the Hamming distance), the design task for a Hopfield associative memory is to find a W so that the given vectors are made into stable fixed points of the net with the size of the attracting basin equal to r. As each neuronal state in the standard Hopfield net is supposed to contribute one entry to the state vector, the total number of neurons in the net (thus the structural extent of the net) is naturally fixed to be the dimension of the given vectors. Therefore, the memory design task actually amounts to finding an appropriate weight matrix W with a *fixed dimension*. Observe that the much-addressed topics, such as stability and reducing the number of spurious states, are only parts of design requirements. A Hopfield associative memory will provide the desired performance if and only if it is designed to meet the specifications of both the stable fixed-points and the basins of attraction.

Recall that the design problem of a Hopfield net associative memory can be formulated in two different ways which are referred to as *structure oriented* and *performance oriented* in Section 3.1.2 of this chapter. The traditional method to design a standard Hopfield associative memory is, as presented above, to set the number of neurons used in the net to be the dimension of the stored patterns and then use the Hebbian rule to set up the connection weights. This structure oriented design method actually provides no mechanism to guarantee the given patterns to be turned into fixed points, let alone the attracting basins. In Section 3.2, a new design method is proposed to design a modified Hopfield net with hidden neurons, which make all the given patterns to be the fixed points of the net. This performance oriented design method of adding hidden

neurons will be pursued even further to incorporate the requirement on the size of attraction basins.

3.3.2 Expanded Hopfield nets

A. **Top-down versus bottom-up design**

Let us first emphasize the essential point of the design of a standard Hopfield net: It starts with a structurally rigid and nonextensible net and tries to store a set of patterns into the model by manipulating a set of internal parameters (weights and thresholds). This is a typical example of the so-called *top-down* design. A more suggestive and widely accepted term is the *model-driven* design reflecting the fact that the model always comes first and data should be stored with the constraints on the model.

Memory systems designed in a top-down fashion can be characterized in terms of the maximal number of data they can actually store without affecting the desired memory performance. This is the *memory capacity*. For the top-down design to be effective, a fundamental requirement is that the memory capacity does not abide, in what-so-ever the way, by any specific nature of the data. Otherwise, there will be a great deal of uncertainty in memory function rendering the designed memory virtually useless. Memory systems satisfying this fundamental requirement are called *general purpose memory systems*.

It is interesting to note that the conventional RAM (random-accessible-memory) represents a distinctive example of the top-down design. A RAM system obviously allows a data-independent memory capacity, as a RAM of r-bit address and k-bit data can store precisely 2^r number of data of dimension k, regardless the nature of these data. Moreover, if we consider r as defining the extent of the RAM structure, the ratio $\rho \triangleq 2^r/r$ measuring the relative capacity of the RAM with respect to its structure is fairly great indeed. In contrast, the top-down designed Hopfield associative memory does not seem to share these advantages. As noted in Chapter 1, if we insist on the memory capacity to be defined in a data-independent fashion, then the capacity will be extremely small. Moreover, the capacity relative to the size of the net remains fixed despite the increase in the number of neurons in the net (see, e.g., [17]).

These deficiencies suggest that it might be appropriate to approach the design of the Hopfield associative memory just the other way round: the

bottom-up (or data-driven) design.

B. **The Structure of the expanded Hopfield net**

Specializing to the design of the Hopfield associative memory, our idea of the bottom-up approach is simply to fix the type of neurons, a state-updating rule, and to propose a design algorithm which determines both the weight matrix W and the number of neurons n in the Hopfield net for a given set of bipolar patterns and a retrieval range. Note one distinct feature of this bottom-up design: the structural extent of the net n enters as a design parameter.

Allowing the number of neurons to be an adjustable parameter amounts to appending an additional fully-connected net to the standard Hopfield net. This modification results in the so-called *extended Hopfield net* as shown in Figure 3.8. Another way to look at the extended Hopfield net is to consider it as a large Hopfield net bipartite into two subnets. Only the state vector of one of the two subnets is relevant, and the state vector of the other subnet does not bear any specific meaning, is thus subject to arbitrary modification. This latter part of the subnet is suggestively called the *hidden structure*. As a result, the function of the neurons in the net is divided into two categories: the output neurons whose states constitute the pattern vector, and the hidden neurons, whose states only have functional significance and provide the design flexibility.

Just to see how the hidden neurons can provide the desired flexibility, let us consider two patterns p^1, p^2 differing by the Hamming distance 1. It has been shown in [10] that p^1, p^2 can not be stored in any standard Hopfield net as fixed points. They can, however, easily be stored as fixed points of an extended Hopfield net with certain number of hidden neurons, if we first append p^1, p^2 respectively, with some q^1, q^2 such that the two prolonged vectors $[p^{1T}\ q^{1T}]^T$ and $[p^{2T}\ q^{2T}]^T$ are orthogonal. This simple example illustrates the essential role played by the hidden structure: It supplements the given patterns so as to make the storage possible.

C. **Dynamics of the extended Hopfield net**

As in the case of the standard Hopfield net, a discrete time scale is considered so that at each time instant, part or all the neurons update their states using the linear thresholding function (see [59][10][9] for discussions). The

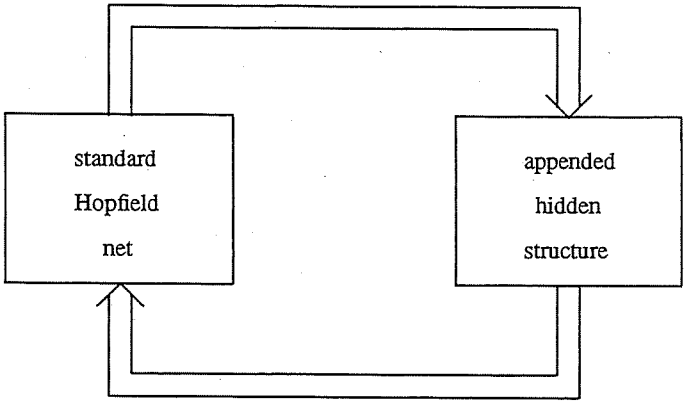


Figure 3.8: The scheme of the extended Hopfield net

asynchronous and the synchronous updating rules represent the two extreme situations in which only 1, respectively, all neurons update their states at each time instant. Note that the asynchronous rule also needs to specify the way by which the updating neuron is picked up.

The standard Hopfield net employs the asynchronous updating rule ([42]). Although this rule can generate a stable discrete dynamical system, it proves to be extremely hard to characterize the precise nature of the attracting basin for each stable fixed-point ([20]). This is, perhaps, the fundamental reason why the design of the Hopfield net does not involve the explicit specification on the basins of attraction.

Since the retrieval range is an indispensable part of the design specification for neural associative memories, we wish to consider the dynamics which allow direct control over the sizes of the attracting basins around each of the stable fixed-points. As will be shown, the synchronous updating rule turns out to define such dynamics. Thus, the extended Hopfield net uses the synchronous state updating rule.

The synchronous updating rule may not always guarantee the stability for

the extended Hopfield net. However, this is not at all essential as long as we can ensure that the desired fixed-points have prescribed attracting basins.

3.3.3 Design rule

A. **Pattern conditions**

To begin with, let us explicitly state our neural associative memory design problem: Given a set of binary patterns $p^1, p^2, \ldots, p^m$ of dimension k and a retrieval range specified by an integer r, we wish to find n and W, the number of neurons, respectively, the connection weight matrix of an extended Hopfield net, such that each of the given patterns will be stable fixed-points of the net with a basin of attraction of size r, *i.e.*, any pattern which has the Hamming distance equal to or less than r from some p^i $(i = 1, 2, \ldots, m)$ will converge to p^i. As our data-driven formulation of the design problem implies, it is the net structure that will be tailored to suit the data specification. Hence, the capacity issue does not arise, and the given set of patterns can be arbitrary in number as well as in nature.

Actually, because of the explicit requirement on the retrieval range, the binary patterns that can be given are not entirely arbitrary. They have to be so specified as to avert contradiction. To be precise, if r (note $r \leq k$, where k is the dimension of the given patterns) is fixed, then the given set of patterns should satisfy

$$p^{iT} \cdot p^j < k - 4r \tag{3.11}$$

for any $i \neq j$ $(i, j = 1, 2, \ldots, m)$, in order to avoid the conflicting storage specification (i.e., one particular input pattern may be required to converge to more than one stable fixed points).

(3.11) follows straight forward from the definition of the Hamming distance of two binary vectors, $\{-1, +1\}$ vectors in our case. Suppose two such vectors p^i and p^j $(i \neq j)$ are apart from each other with a Hamming distance h, then the inner product of these two k-dimensional $\{-1, +1\}$ vectors is given as follows:

$$p_i^T \cdot p_j = k - 2h. \tag{3.12}$$

Recall that the integer r has been defined as the size of the attracting basins of pattern p^i and p^j. It is apparent that the following should hold

$$h > 2r \tag{3.13}$$

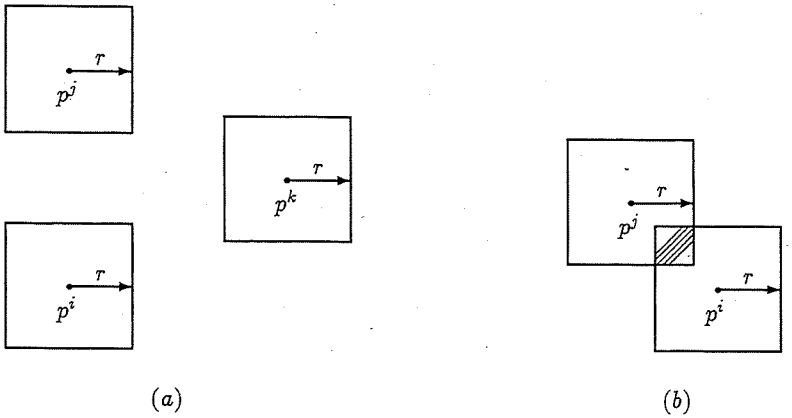


Figure 3.9: An illustration of the pattern condition: (a). A legitimate pattern specification; (b). A contradictory specification.

in order for p^i and p^j to have prescribed non-overlapping attracting basins. Substitute inequality (3.13) into Equation (3.12), we can have inequality (3.11). Subsequently, the pattern condition (3.11) must be satisfied to avoid overlapping of different attracting basins.

To visualize the condition (3.11), let us conceive each p^i as the center of the ith square with the side length r in the 2-dimensional plane. The m patterns will correspond to m such squares. (3.11) then requires that these m squares will not overlap (see Figure 3.9). The arbitrariness of the patterns is therefore confined to within the condition (3.11).

Observe that the retrieval range (or the size of attracting basins) r also poses a limitation on m, the number of given patterns. In the two extreme cases, m can either be 1 or 2^k (the maximal number of k-dimensional binary patterns), depending on whether r is k or 0.

Given the retrieval range r ($0 \leq r \leq k$), the number of patterns that fall within the attracting basin of a particular stable fixed point $N_i (i = 1, 2, \cdots, m)$

can be calculated in a combinatorial way as follows:

$$N_i = \sum_{i=0}^{r} \binom{k}{i}. \tag{3.14}$$

Since every stable fixed point has the same size of attracting basin, hence the N_i's are the same for all the m fixed points. Further since total number of the binary patterns in a k-dimensional space is 2^k, so we have

$$m \cdot N_i \leq 2^k,$$

or equivalent

$$m \cdot (\sum_{i=0}^{r} \binom{k}{i}) \leq 2^k, \tag{3.15}$$

which indicates that if k and r are fixed, we are only free to specify at most a γ number of patterns (stable fixed points) for the memory storage without leading to conflicting requirement, where γ is the maximal m in (3.15)

$$\gamma = \frac{2^k}{\sum_{i=0}^{r} \binom{k}{i}}.$$

The coding nature of the binary patterns naturally implies another restriction on the patterns: any pattern p is not distinguishable from its negative, $-p$. An intuitive justification for this is that if p is visualized as a binary picture of white foreground black background, then $-p$ is just the other way round. Thus, the two patterns code the same information. To pursue this consideration further, if a p' is within the retrieval range of p, then $-p'$ should be within that of $-p$. The maximum retrieval range for each distinct pattern is therefore reduced to $\lfloor n/2 \rfloor$. Moreover, the upper-bound on m is also reduced by half giving rise to

$$m_{\max} = \frac{2^{k-1}}{\sum_{i=0}^{r} \binom{k}{i}}. \tag{3.16}$$

A set of binary patterns specified for the memory storage under this restriction will be called a *strict set* and it can be strictly defined as follows:

Definition 3 *A strict set with r as the size of attracting basins is defined to be a set of k-dimensional binary patterns that have the following features: It at most has m_{max} patterns; it will not simultaneously contain a pattern p and its reciprocal $-p$; finally, the Hamming distance between any two patterns in the set will be between $2r$ and $\lfloor n/2 \rfloor$.*

B. Orthogonalization theorem and the storage rule

If it so happens that the given patterns are mutually orthogonal, *i.e.*, ${p^i}^T p^j = k\delta_{ij}$ (δ_{ij} is the Kronecker function which is 1 when $i = j$, 0 otherwise), then we simply set n, the number of neurons in the net, to be k, the dimension of the given patterns, and then use the well-known sum-of-outer product rule to determine the weight matrix W

$$W = \sum_{i=1}^{m} p^i {p^i}^T. \tag{3.17}$$

Note that this time the diagonal entries of W will not be set to zero (compare this with (1.1)). To justify this simple rule, let us observe that for any p^j ($j = 1, 2, \ldots, m$)

$$\text{Sign}(Wp^j) = \text{Sign}(\sum_{i=1}^{m} p^i {p^i}^T p^j) = \text{Sign}(p^j {p^j}^T p^j) = \text{sign}(kp^j) = p^j,$$

hence every p^j is a fixed point of the net.

This special case motivates an orthogonalization design approach. The basic idea is that if the set of given patterns are not mutually orthogonal, we may first orthogonalize them by appending to each of the given patterns a number of appropriate binary elements to form a new set of orthogonal patterns with a high dimension, then apply the design rule (3.17) to find the weight matrix for the expanded Hopfield net.

To be precise, let all the given patterns be arranged into a matrix A

$$A = [p^1 \;\; p^2 \;\; \cdots p^m].$$

We first construct a binary element matrix B of dimension $s \times m$

$$B = [q^1 \;\; q^2 \;\; \cdots q^m]$$

where q^i ($i = 1, 2, \cdots, m$) is an s-dimensional binary vector, such that the matrix $\begin{bmatrix} A \\ B \end{bmatrix}$ has mutually orthogonal columns. We then design the extended Hopfield net to have $k + s$ number of neurons with the weight matrix given by

$$W = \begin{bmatrix} A \\ B \end{bmatrix} [A^T \;\; B^T]. \tag{3.18}$$

The whole net can be seen as consisting of two subnets with the respective weight matrices AA^T and BB^T, and the two subnets are connected by AB^T and BA^T. Clearly, any column vector $\begin{bmatrix} p^j \\ q^j \end{bmatrix}$ is a fixed-point of the net and part of the fixed-point p^j ($j = 1, 2, \cdots, m$) is what we need.

For such a design approach to work, we have to be sure that B exists for any A. The following theorem provides a constructive proof for the existence of B.

Theorem 3 *For any binary matrix A of dimension $k \times m$, there exists another binary matrix B of dimension $s \times m$ (for certain integer s) such that the columns of the matrix $\begin{bmatrix} A \\ B \end{bmatrix}$ are mutually orthogonal.*

Proof:

Let $p^1, p^2, \ldots, p^m$ be the columns of the matrix A. For the ease of notation, we denote by A_i the matrix composed of the first i columns of the matrix A, thus i goes from 1 to m. The proof is by induction.

- For $i = 1$, A_1 becomes a column vector. There is nothing to prove ($B_1 = 0$);
- For $i = 2$, $A_2 = [p^1 \;\; p^2]$. Let $B_2 = [p^1 \;\; -p^2]$, we can obtain a new matrix C_2,

  $$C_2 = \begin{bmatrix} A_2 \\ B_2 \end{bmatrix} = \begin{bmatrix} p^1 & p^2 \\ p^1 & -p^2 \end{bmatrix}.$$

  It can be easily verified that the matrix C_2 has mutually orthogonal columns;

- Suppose that for certain i $(2 < i \leq m)$, there exists a matrix B_i such that a new matrix C_i

$$C_i = \begin{bmatrix} A_i \\ B_i \end{bmatrix}$$

has orthogonal columns;

- By definition $A_{i+1} = [A_i \quad p^{i+1}]$. Construct the matrix B_{i+1} in the following way:

$$B_{i+1} = \begin{bmatrix} B_i & P^{i+1} \\ A_i & -p^{i+1} \\ B_i & -P^{i+1} \end{bmatrix},$$

where P^{i+1} has the same number of rows with B_i and

$$P^{i+1} = \begin{bmatrix} p^{i+1} \\ p^{i+1} \\ \vdots \\ p^{i+1} \end{bmatrix},$$

then it is easy to verify that the new matrix

$$C_{i+1} = \begin{bmatrix} A_{i+1} \\ B_{i+1} \end{bmatrix} = \begin{bmatrix} A_i & p^{i+1} \\ B_i & P^{i+1} \\ A_i & -p^{i+1} \\ B_i & -P^{i+1} \end{bmatrix}$$

have mutually orthogonal columns.

The induction step is thus complete. The final matrix B follows from the construction. The theorem is thus proven. □

Note that Theorem 3 only provides a rather general way to construct the B matrix. Other special methods for some special matrix A may also be applied resulting in a relatively small s (the number of rows in B matrix). This is especially the case when the columns of A have some regular structures (e.g., some of the columns of A matrix are already mutually orthogonal).

3.3.4 Relations between attracting basins and the network structure

The design rule (3.18) given in the preceding subsection only ensures that the given patterns are made the fixed-points of the net. It does not automatically guarantee the prescribed attracting basin for these fixed-points without further modification. The following theorem develops a sufficient condition for the prescribed basin to exist. This result is entirely based on the synchronous updating assumption of the extended Hopfield net. We have also assumed a stronger convergence mode, one-step convergence, so that the desired fixed-point is reached in just a single time step for any initial pattern falling into its attracting basin.

Theorem 4 *The condition $n > 2rm$ $(n = k + s)$ is sufficient to ensure that (3.18) can store following m mutually orthogonal patterns:*

$$\begin{bmatrix} p^1 \\ q^1 \end{bmatrix}, \begin{bmatrix} p^2 \\ q^2 \end{bmatrix}, \cdots, \begin{bmatrix} p^m \\ q^m \end{bmatrix}$$

as stable fixed-points of an n-neuron Hopfield net, each of which has an attracting basin of size r.

Proof:

For simplicity of the notation let $\begin{bmatrix} p^i \\ q^i \end{bmatrix}$ be denoted by a_i $(i = 1, 2, \ldots, m)$. Then $a_1, a_2, \cdots, a_m$ are n-dimensional mutually orthogonal binary column vectors satisfying

$$a_i{}^T a_j = n\delta_{ij},$$

where $\delta_{ij} = \begin{cases} 1 & \text{when } i = j \\ 0 & \text{otherwise} \end{cases}$. The extended Hopfield net is defined by setting the number of neurons to be n, and the weight matrix W to be

$$W = \sum_{i=1}^{m} a_i a_i^T.$$

Let us assume that $\tilde{a}_t$, for some $t \in \{1, 2, \ldots, m\}$, be given as the initial pattern of the net, which has a Hamming distance smaller than or equal to r

from a stored pattern a_t, thus the following inequality about the inner product of these two vectors should hold

$$a_t^T \tilde{a}_t \geq n - 2r \tag{3.19}$$

From the pattern condition discussed in a previous subsection, $\tilde{a}_t$ should not fall in the attracting basin of any other fixed point a_i $(i \neq t,\ i = 1, 2, \cdots, m)$, or say it in another word, the Hamming distance between $\tilde{a}_t$ and a_i $(i \neq t)$ should be larger than r, hence we have the following

$$|a_i^T \tilde{a}_t| \leq 2r \quad \text{for } i \neq t. \tag{3.20}$$

As we assume the synchronous updating rule, the following product is formed during the state updating of the net

$$W\tilde{a}_t = \sum_{i=1}^{m} a_i a_i^T \tilde{a}_t. \tag{3.21}$$

It follows that if the one-step convergence is required, the thresholding of the term on the right hand side of (3.21) should be a_t. From (3.21), we have

$$W\tilde{a}_t = (a_t^T \tilde{a}_t)a_t + \sum_{i \neq t}^{m} (a_i^T \tilde{a}_t)a_i. \tag{3.22}$$

For the thresholding of the term on the right hand side of (3.22) to be a_t, the worst case is that all the a_i's $(i \neq t)$ are equal to $-a_t$. So, for the worst case, (3.22) becomes

$$W\tilde{a}_t = (a_t^T \tilde{a}_t - \sum_{i \neq t}^{m} a_i^T \tilde{a}_t)a_t. \tag{3.23}$$

In order for $\text{Sign}(W\tilde{a}_t) = a_t$, it is sufficient that

$$a_t^T \tilde{a}_t - \sum_{i \neq t}^{m} a_i^T \tilde{a}_t > 0. \tag{3.24}$$

Substitute (3.19) into (3.24), we have

$$(n - 2r) - 2r(m - 1) > 0,$$

or equivalently,

$$n > 2rm. \tag{3.25}$$

The theorem is thus proven. □

The theorem tells that the number of neurons in the net should be sufficiently large to satisfy the basin requirement. As the number of neurons in the extended Hopfield net is determined by $n = k + s$ (recall that s is the number of rows in the matrix B defined in (3.18)), the condition (3.25) states that s should be greater than $2rm - k$ in order to ensure the attracting basin of size r. It is easy to realize that s obtained as a result of constructing the matrix B in the preceding section may not always satisfy this condition. If this is the case, we can always find another $(s' \times m)$-matrix B' with orthogonal columns so that $s + s' > 2rm - k$. Hence, $n\ (= k + s + s')$ will be the number of neurons in the new net with the weight matrix defined by

$$W = \begin{bmatrix} A \\ B \\ B' \end{bmatrix} [A^T \ \ B^T \ \ B'^T].$$

It should also be observed that (3.25) is derived for the worst possible case. For the actual set of given patterns, the convergence may still happen even without satisfying this condition.

3.3.5 Conclusions

In this section, an effort is made to reformulate the neural associative memory design problem. In contrast to the original probabilistic and model-driven formulation, the new framework is deterministic and data-driven in nature. The advantage of the present formulation is that it will always lead to a finite net structure with the required stable fixed points and the prescribed basin of attraction for *any* set of patterns satisfying the pattern condition. This is, however, not always the case in the original formulation. This bottom-up design also avoids troublesome issues such as capacity.

An important issue which is inherent in the extended Hopfield net is that for the actual operation of the net, we need to specify the initial condition for the hidden structure of the net as well. This additional requirement amounts

to supplementing the given distorted pattern with a context information as discussed in [51]. Currently, we are generating this context information from an external computation. It would be interesting if the generation is done purely internally. But this would probably require a modification to the proposed net structure. This and related issues deserve further clarification.

3.4 An optimization based design rule

3.4.1 Introduction

It has been emphasized in the previous sections that a standard Hopfield associative memory designed using the Hebbian rule does not guarantee all the given patterns to be its stable fixed points with certain size of attraction basins. It is also known that only when the given set of patterns possess some particular internal structure (e.g., mutually orthogonal) or the size of the net is stretched to the infinite, the designed memory can function as desired.

To design practically usable neural associative memories, the problem has been reformulated in two ways in Section 3.1. Under the first formulation, two methods have been presented in Section 3.2 and Section 3.3 to solve the problems associated with the traditional Hopfield associative memories from different aspects. One common drawback of these two methods is that it is not economical from the point of view of the number of the neurons used in the net. Since each neuronal state in a neural associative memory is supposed to contribute one entry to the state vector, the smallest number of neurons used in the net is apparently the dimension of the given patterns. It follows that when the number of the neurons used in the net rises to be one of the main considerations, the design problem of a Hopfield associative memory should be formulated in the second way, i.e.: Given a set of n-dimensional bipolar vectors $f^1, f^2, \ldots, f^m$, and assuming the number of the neurons to be used in the net equal to n, find an $n \times n$ weight matrix W, so that $f^1, f^2, \ldots, f^m$ are stable fixed-points of the net with the maximal possible attraction basin. Note that the Hebbian rule can be considered into this formulation except that it does not explicitly introduce the fixed points and size of attraction basin as design objectives.

The objective of this section is to provide a new method of designing the

Hopfield associative memory under the second formulation. As will be seen in the following, the most important feature of this formulation is the explicit introduction of the size of attraction basin to be a main design parameter, and the connection matrix is obtained as a result of optimizing such a parameter. A particular feature of this method is that all the connection weights can only assume three different values, -1, $+1$ and 0, which facilitates the VLSI implementation of the weights. Compared to the Hebbian rule, the new method can guarantee all given patterns to be stored at least as fixed points, regardless the internal structure of these patterns. The new design method is illustrated by a few examples, and the performance of the obtained networks are compared with those obtained using the Hebbian rule. The results indicate that the approach out-performs the Hebbian rule when the given patterns do not possess any particular structure and is as good as it when the patterns are mutually orthogonal.

This section is structured as follows. To fix the idea, some definitions concerning the convergence of the Hopfield net and a few conventions are introduced in the subsequent subsection. In Subsection 3.4.3, we present the new approach to design the Hopfield associative memory and one of the algorithms to implement it. Some design examples are given and the results are compared to the Hebbian rule in Subsection 3.4.4. Conclusions and some additional remarks are included in Subsection 3.4.5.

3.4.2 Definitions and conventions

A. **Types of convergence**

Before proceeding, let us first clarify the various definitions on the basins of attraction corresponding to different convergence behaviors of the Hopfield net. We shall only be concerned with the discrete *asynchronous* Hopfield net.

As is defined in the previous subsection, every possible state of the Hopfield net with n neurons, will necessarily be an n-dimensional $\{+1, -1\}$ vector, corresponding to one of the 2^n vertices of an n-dimensional hypercube. A vertex x is said to be a stable fixed point if it is invariant after an arbitrary single-step state-update, and if there exist other vertices $y_1, y_2, \cdots (y_i \neq x)$ such that they converge to x after finite steps of state-updates. All the vertices $y_1, y_2, \cdots$, are said to belong to the basin of attraction of x. Intuitively, we may expect that the dynamics of the net makes the vertices closer (in the sense

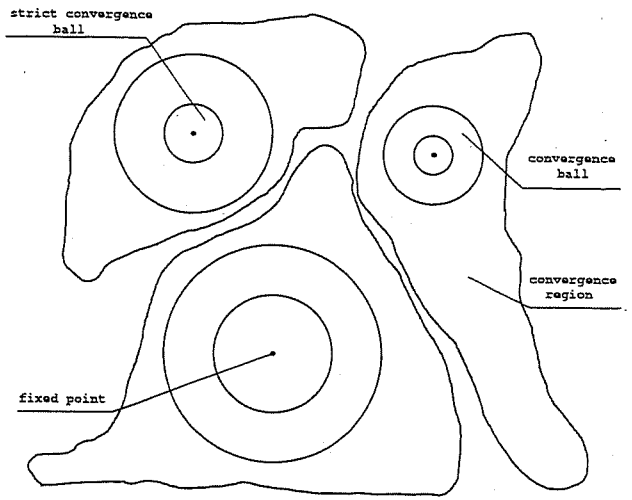


Figure 3.10: Three different types of attracting basins and fixed points.

of Hamming distance) to a stable fixed point converge to it, so that the basin of attraction assume regular shapes, like, e.g., a ball. Unfortunately, this is not always the case as is exemplified in [59].

Here we identify some special types of convergence behaviors below. Remind that the following definitions assume asynchronous state update.

Definition 4 *The convergence region of a stable fixed vertex is defined to consist of all the vertices which converge to the fixed vertex in a finite number of steps of state updating.*

Definition 5 *The convergence ball of a stable fixed vertex is a convergence region shaped as a ball, i.e., it consists of all the vertices with their Hamming distances to the stable fixed vertex less than some constant integer.*

Definition 6 *The strict convergence ball of a stable fixed vertex is a convergence ball of the vertex in which all the vertices will converge to that fixed vertex in such a way that they will never increase the Hamming distance from the stable fixed vertex during each step of state updating.*

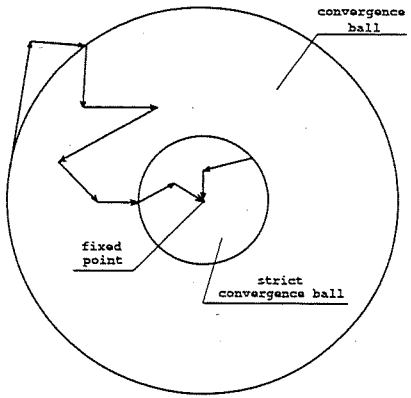


Figure 3.11: The different convergent behaviors within convergence ball and strict convergence ball.

Here we emphasize that the convergence in the preceding definitions should be understood to be a result of asynchronous state-update. We shall have more on this in the ensuing subsection.

The relations among these three definitions are roughly illustrated in two dimensional space in Figure 3.10. Observe the difference between the *convergence ball* and the *strict convergence ball* of a vertex. For a *convergence ball*, a vertex starting inside the ball may first run out of the ball after certain step of state-update, then converge back to the fixed point, while for a *strict convergence ball*, any vertex starting inside the ball will go strictly towards the fixed point. This difference is demonstrated in Figure 3.11.

The types of attraction basin we are concerned with here will be confined to the *strict convergence ball*. Obviously, the *strict convergence ball* is the smallest among the three definitions of the attracting basin. We choose this because of the fact that it is simple to analyze and easy to control in the design

process.

B. **Asynchronous and synchronous state-update**

The dynamical behaviour of a net can only be fixed if we specify a definite way to update the state of the net. There exist two different types of state updating rules: synchronous and asynchronous rules. The former updates the entire state vector at each time instance; the latter updates only one entry of the state vector chosen according to certain scheme (orderly, or random) at each time instance. The following theorem exhibits the relationship between the two state-updating rules. This relationship implies that it is basically the same (under some mild assumptions) to either deal with a single-step convergence under the synchronous updating, or a multi-step convergence under any asynchronous updating, if the underlying attraction basin is a *strict convergence ball.*

Theorem 5 *The sufficient and necessary condition for a set of vertices to be in the strict convergence ball of a fixed point is that starting from any vertex in the set it will converge to the fixed point in one step, if the state vector of the same net is updated synchronously.*

Proof:

Denote a stable fixed vertex by

$$f = [f_1, f_2, \ldots, f_{i_1}, \ldots, f_{i_2}, \ldots, f_{i_r}, \ldots, f_n]^T$$

and a vertex in a set Ω by

$$p = [f_1, f_2, \ldots, p_{i_1}, \ldots, p_{i_2}, \ldots, p_{i_r}, \ldots, f_n]^T,$$

where $f_{i_j} \neq p_{i_j}$ for $j = 1, 2, \ldots, r$. Further, the connection weight matrix of dimension $n \times n$ is partitioned as

$$W = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix},$$

where w_i $(i = 1, 2, \ldots, n)$ is an n-dimensional row vector representing all the weights connecting to the neuron i.

Sufficiency:

Suppose that the net with initial state p can converge to the fixed point f in one step using the synchronous updating rule, i.e.,

$$\text{Sign}(W \cdot p - \theta) = f, \tag{3.26}$$

or equivalently,

$$\text{Sign}(w_i \cdot p - \theta_i) = f_i \quad i = 1, 2, \ldots, i_1, \ldots, i_2, \ldots, i_r, \ldots, n \tag{3.27}$$

Now let the same net undergo the asynchronous state-update. From (3.27) we know that for any j, $j \in (i_1, i_2, \cdots, i_r)$, the state of the jth neuron will either flip or remain the same after one such update, whereas the states of other neurons will remain unchanged. It therefore follows that the Hamming distance between the initial net state p and the new net state after the update will decrease by 1 or remain the same.

Because the vertex p is arbitrarily chosen from the set Ω, the proof is in fact true for all the vertices in Ω. Thus Ω is a *strict convergence ball* for f.

Necessity:

Suppose that the set Ω is a *strict convergence ball* of certain f, it follows that starting from any vertex p in the set, the net will change (if it changes at all) its state towards the fixed point f after one step asynchronous state update for neuron j. More precisely, when $j \in \{i_1, i_2, \ldots, i_r\}$, the next state of the jth neuron will change from p_j to f_j (p_j, f_j are, respectively the jth entries of the vectors p and f); when $j \notin \{i_1, i_2, \ldots, i_r\}$, the state of the jth neuron will still be f_j. Consequently, we have the following equations:

$$\text{Sign}(w_i \cdot p - \theta_i) = f_i \quad i = 1, 2, \ldots, i_1, \ldots, i_2, \ldots, i_r, \ldots, n, \tag{3.28}$$

or in matrix form,

$$\text{Sign}(W \cdot p - \theta) = f. \tag{3.29}$$

(3.29) shows that p will converge to f after a single-step synchronous state update.

The proof is thus completed. □

3.4.3 The new design rule

Now we are in a position to develop the new design approach. As is already made clear in the preceding sections, for a set S of m n-dimensional binary patterns,

$$f^1 = [f^1{}_1\ f^1{}_2\ \cdots\ f^1{}_n]^T$$
$$f^2 = [f^2{}_1\ f^2{}_2\ \cdots\ f^2{}_n]^T$$
$$\cdots$$
$$f^m = [f^m{}_1\ f^m{}_2\ \cdots\ f^m{}_n]^T,$$

the goal is to design an n-neuron discrete Hopfield net in such a way that all these patterns be made the stable fixed points of the net with as large as possible strict convergence balls surrounding each of the fixed points.

Let the weight matrix be given by $W = [w_1{}^T w_2{}^T \cdots w_n{}^T]^T$, in which the n-dimensional row vector w_i $(i = 1, 2, \ldots, n)$ represents all the weights connecting to the ith neuron. Without symmetric assumption on the weight matrix W, each row w_i $(i = 1, 2, \cdots, n)$ of W is not structurally related to the other rows, and can be handled separately. We proceed by partitioning S into two subsets, S_i^- and S_i^+, in the following fashion: All the patterns with the ith entry being -1 are set in S_i^- and those with $+1$ are set in S_i^+. Note that S_i^- and S_i^+ are different for different i although for any i, $S_i^- \cup S_i^+ = S$. Let the activation of the ith neuron with respect to a pattern f be defined by $w_i \cdot f$, the weighted sum of all the inputs to this neuron. Then by taking each pattern in S_i^- as an input pattern, we obtain a set of activation values corresponding to the ith neuron and the maximum among these activation, denoted by $v_i\bar{\max}$, can easily be identified. Similarly, we obtain the minimum, denoted by $v_i\bar{\min}$, among all the activation values with respect to all the patterns in S_i^+. Based on $v_i\bar{\max}$ and $v_i\bar{\min}$, we have the following definition:

Definition 7 *The activation difference d_i for the ith neuron is defined to be*

$$d_i = v_i\bar{min} - v_i\bar{max} \quad i = 1, 2, \cdots, n. \tag{3.30}$$

Clearly, d_i can be different for each neuron. It is generally true that $d_i \neq d_j$ for $i \neq j$, and further, any change of d_i due to the change of the weight vector w_i will have no effect on d_j $(j \neq i)$. In this sense, we say that

all the *activation differences* are mutually independent. It then follows that each d_i can be adjusted as desired by adjusting w_i separately. Next, we fix the threshold value for the ith neuron θ_i to be the following:

$$\theta_i = \frac{v_i\bar{\max} + v_i\bar{\min}}{2}. \tag{3.31}$$

On choosing this particular threshold value, and denoting by $d_{\min}$ the smallest activation difference among d_i's, and by $r_{\min}$ the smallest possible *strict convergence balls* for all the given patterns, we can establish the following interesting relationship between $d_{\min}$ and $r_{\min}$.

Theorem 6 *For a discrete Hopfield associative memory, the radius of the smallest strict convergence ball r_{min} and the smallest activation difference d_{min} has the following simple relation:*

$$r_{min} = \lfloor \frac{d_{min} - 1}{4} \rfloor, \tag{3.32}$$

where $\lfloor \cdot \rfloor$ is the floor function.

Proof:

Suppose that the ith neuron attains the smallest *activation difference* $d_{\min}$. Since the weights only assume three different values ± 1 and 0, the flip of any entry in the input pattern (either $-1 \longrightarrow +1$, or $+1 \longrightarrow -1$) can at most contribute ± 2 to the the activation of the neuron. More precisely, let us assume that the ith neuron attains $v_i\bar{\max}$ when the input pattern is p^-, and attains $v_i\bar{\min}$ when the input pattern is p^+. If one of the entries of p^- flips (either $-1 \longrightarrow +1$ or $+1 \longrightarrow -1$), it will lead to an increase Δv of either 0, ± 1 or ± 2 in the value of $v_i\bar{\max}$, with the greatest gain being $\Delta v = +2$. In a similar fashion, when h number of entries in p^- are reversed leading to a pattern p_h^- (with Hamming distance h to p^-) taken to be an input pattern, the greatest gain in $v_i\bar{\max}$ is $+2h$. It is easy to see that if

$$v_i\bar{\max} + 2h - \theta_i < 0 \tag{3.33}$$

still holds, p_h^- will converge to p^- in one synchronous updating (or equivalently, p_h^- is in the *strict convergence ball* of p^-). Upon substituting θ_i in (3.33) by (3.30) and (3.31) we obtain

$$2h < d_{\min}/2.$$

Thus the maximum of such h should satisfy

$$h_{\max} = \lfloor \frac{d_{\min} - 1}{4} \rfloor. \tag{3.34}$$

As $h_{\max}$ turns out to be $r_{\min}$, the theorem is thus proven. □

The theorem shows that the size of the minimal *strict convergence ball* is directly linked to the parameter $d_{\min}$ which is in turn determined by the weight matrix W. Thus, for a given W, $d_{\min}$ can be used to evaluate the minimal possible basin of attraction, hence the effectiveness of the design scheme. As will be seen in the next section, for some pattern sets, W constructed using the Hebbian rule will result in $d_{\min}$ being negative, indicating not only the absence of the basin of attraction around some given patterns, but also the fact that some patterns can not even be made into the fixed points.

The theorem also suggests that to maximize $r_{\min}$ as part of the design specification, we can as well maximize the parameter $d_{\min}$. Following this general idea, we can simply start with an arbitrary W_0, and propose a weight updating scheme so that after a finite number of such updates, $d_{\min}$ attains its maximum.

As the entries of the weight matrix are confined to be 0 and ± 1, the weight matrix itself can assume up to 3^{n^2} number of different configurations. Thus at the extreme, all the different weight matrices can be exhaustively tried out. But this is only feasible when n is small. In general, any weight updating scheme will incorporate, in one way or another, a sort of search mechanism in order to find the maximal $d_{\min}$ at just few evaluations of different weight matrix configuration.

The following is one possible scheme that we have used to maximize $d_{\min}$.

Algorithm:

Initialization: Set W to be an arbitrary matrix with entries being 0, ± 1. Preferably, W can be obtained by the Hebbian rule for a given set of patterns.

For $i := 1$ **To** n

Step 1 Divide the patterns into two subsets S_i^- and S_i^+;

Step 2 Compute the *activation difference* d_i using the current w_i;

Step 3 Randomly choose one of the entries of w_i, e.g., w_{ij};

Step 4 Update w_{ij} in such a way that the change of w_{ij} will increase d_i (e.g., using the gradient method);

Step 5 If no increase is possible, and if the number of random selection as in **Step 3** is less than n, go to **Step 3**; Otherwise stop.

Although we can experimentally demonstrate that this algorithm can find the optimal W so that $d_{\min}$ reaches its maximum when n is small, the theoretical proof is yet to be worked out.

3.4.4 Illustrative examples

In the following examples, we will store each set of the 20-dimensional patterns in a 20-neuron discrete Hopfield net as stable fixed points with as large as possible *strict convergence ball*. One of the examples has been taken from [49], but the design in this reference seems to be faulty. The advantages of our approach can clearly be seen from the simulation results. It is especially superior to the Hebbian rule when the given patterns do not possess any particular internal structure (e.g., mutual orthogonality).

Example 1 (Randomly generated patterns):

Pattern 1: +1 −1 −1 −1 +1 +1 +1 +1 −1 +1 +1 −1 −1 +1 −1 −1 +1 −1 +1 +1
Pattern 2: +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 −1 +1 +1 −1 +1 +1
Pattern 3: +1 +1 +1 −1 +1 +1 +1 +1 −1 +1 +1 +1 −1 −1 +1 +1 +1 +1 +1 +1
Pattern 4: +1 +1 +1 +1 +1 +1 +1 −1 +1 +1 +1 +1 +1 +1 +1 −1 +1 +1 +1 −1
Pattern 5: +1 +1 −1 +1 +1 +1 +1 −1 +1 −1 +1 +1 −1 +1 −1 +1 −1 +1 +1 +1

Two Hopfield nets are designed one with the Hebbian rule and one with our approach, respectively, we summarize the computed parameters in the following table:

neuron No. i		1	2	3	4	5	6	7	8	9	10
d_i	Hebb rule	*	10	8	8	*	*	*	8	8	10
	our method	*	14	8	10	*	*	*	10	10	12
τmin	Hebb rule	0									
	our method	1									

continue the above table

neuron No. i		11	12	13	14	15	16	17	18	19	20
d_i	Hebb rule	*	10	8	6	8	6	10	4	*	6
	our method	*	14	8	12	8	6	12	8	*	10
τmin	Hebb	0									
	ours	1									

The "*" in the table indicates very large *activation differences* of the corresponding neuron for all the input patterns. This is true because in this case one of the two subsets, e.g., S_i^- (S_i^+) is null set and we can always set the threshold θ_i to a very negative (or positive) value which will drive $d_i = 2(v_i^+{}_{\min} - \theta_i)$ to a very large value. τmin in the table are obtained using (3.32) for both the Hebbian rule and our approach. The table clearly shows that compared with the Hebbian rule, our approach ends up with larger *activation differences* for most of the neurons, extending the *strict convergence ball* by Hamming distance 1.

Example 2 (Mutually orthogonal patterns):
Pattern 1: +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1
Pattern 2: +1 +1 +1 +1 +1 −1 −1 −1 −1 −1 +1 +1 +1 +1 +1 −1 −1 −1 −1 −1
Pattern 3: +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
Pattern 4: +1 +1 −1 −1 −1 −1 −1 +1 +1 +1 +1 +1 +1 −1 −1 −1 −1 −1 +1 +1
Pattern 5: −1 +1 −1 +1 −1 +1 −1 +1 −1 +1 −1 +1 −1 +1 −1 +1 −1 +1 −1 +1

The performances of the nets designed with the Hebbian rule and our approach are compared in the following table:

neuron No. i		1	2	3	4	5	6	7	8	9	10
d_i	Hebb rule	16	*	12	12	12	16	12	16	12	16
	our method	20	*	12	20	12	16	12	20	12	20
τmin	Hebb rule	2									
	our method	2									

continue the above table

neuron No. i		11	12	13	14	15	16	17	18	19	20
d_i	Hebb rule	12	12	12	12	16	12	12	12	12	16
	our method	12	20	12	12	16	12	20	12	12	16
τmin	Hebb rule	2									
	our method	2									

The example shows that for an orthogonal set of patterns, the Hebbian rule is as good as our method.

Example 3 (5 patterns with mutual Hamming distance of 8):
Pattern 1: +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 −1 −1 −1 −1
Pattern 2: −1 −1 −1 −1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1
Pattern 3: +1 +1 +1 +1 −1 −1 −1 −1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1
Pattern 4: +1 +1 +1 +1 +1 +1 +1 +1 −1 −1 −1 −1 +1 +1 +1 +1 +1 +1 +1 +1
Pattern 5: +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 −1 −1 −1 −1 +1 +1 +1 +1

The weight matrix obtained using our approach is:

$W =$

$$\begin{bmatrix}
1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1
\end{bmatrix}$$

The results are shown in the following table:

neuron No. i		1	2	3	4	5	6	7	8	9	10
d_i	Hebb rule	0	0	0	0	0	0	0	0	0	0
	our method	16	16	16	16	16	16	16	16	16	16
$r$min	Hebb rule	$-$ ($d_{\min} = 0$, not all patterns are fixed points)									
	our method	3									

continue the above table

neuron No. i		11	12	13	14	15	16	17	18	19	20
d_i	Hebb rule	0	0	0	0	0	0	0	0	0	0
	our method	16	16	16	16	16	16	16	16	16	16
$r$min	Hebb rule	$-$ ($d_{\min} = 0$, not all patterns are fixed points)									
	our method	3									

This result indicates that for some given sets of patterns, our approach can have a great advantage over the Hebbian rule in terms of making the given patterns to be the fixed points, of allowing maximal possible size of attraction basins under the super-imposed structure.

In addition to the above illustrative examples, more simulations of large scale have been done to test the new approach and to compare with the Hebbian rule. In these simulations discrete Hopfield models of 32, 64 and 128 neurons are used to store different sets (with number of patterns in the set ranging from 2 to 16, 2 to 22 and 2 to 34 respectively) of patterns of dimension 32, 64 and 128. Each set of patterns are randomly generated deterministic bipolar vectors. Based on each set of patterns weight matrices are computed using both the clipped Hebbian rule and our approach. The radii of the smallest *strict convergence balls* are then calculated for the two methods and the results are shown in Figure 3.12, 3.13 and 3.14. Also shown in the figures are the radii in the so called "ideal case", which is actually equal to $\lfloor \frac{H_{\min}}{2} \rfloor$ where $H_{\min}$ is the smallest Hamming distance between the two closest patterns (in Hamming sense) of the set. These radii of "ideal case" can also be looked as an upper bound for any design method of discrete Hopfield associative memory.

To compare the capacities of the discrete Hopfield associative memories designed using the Hebbian rule and our new method, another set of simulations

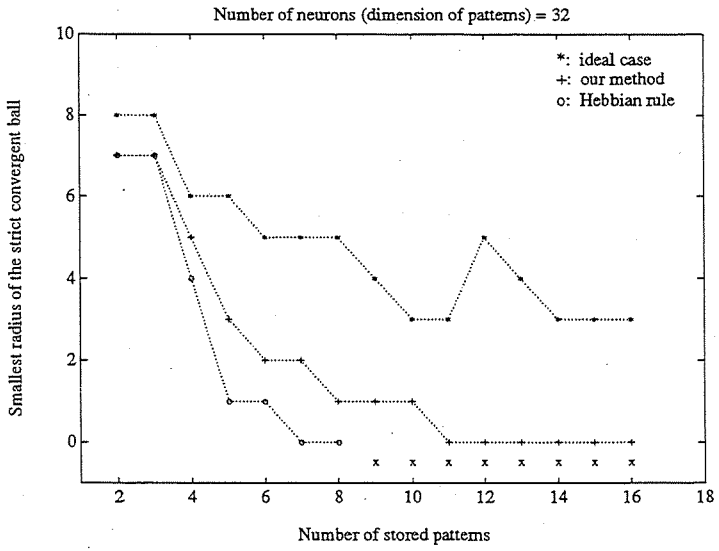


Figure 3.12: Radii of the *strict convergence ball* of 32-neuron Hopfield net obtained with different methods. "x" denotes that not all of the given patterns can be made fixed points by Hebbian rule.

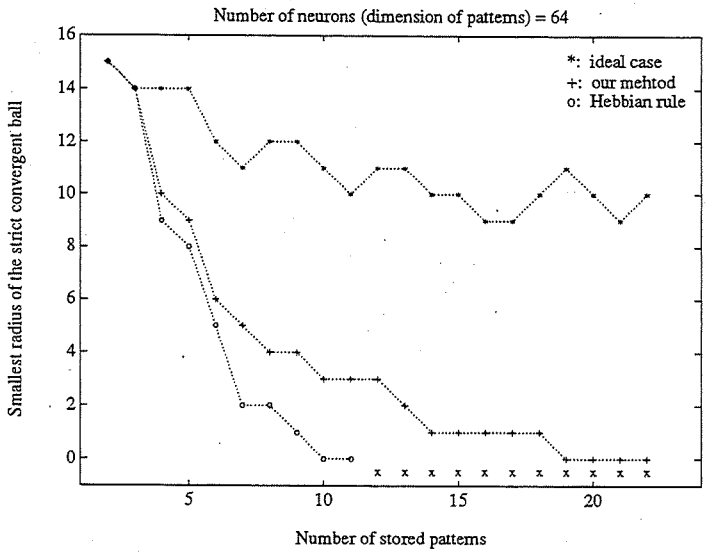


Figure 3.13: Radii of the *strict convergence ball* of 64-neuron Hopfield net obtained with different methods. "x" denotes that not all of the given patterns can be made fixed points by Hebbian rule.

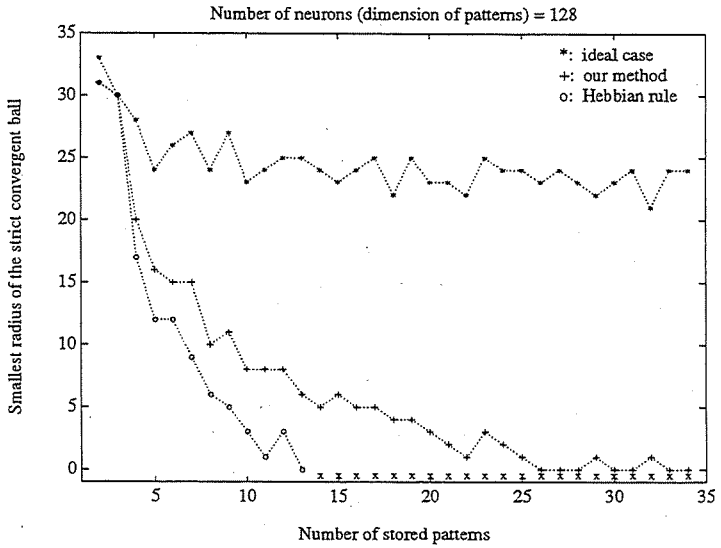


Figure 3.14: Radii of the *strict convergence ball* of 128-neuron Hopfield net obtained with different methods. "x" denotes that not all of the given patterns can be made fixed points by Hebbian rule.

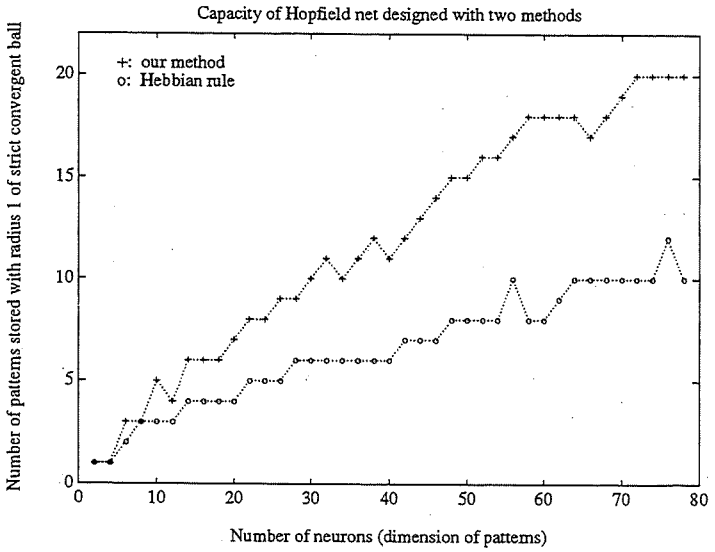


Figure 3.15: Number of patterns that can be stored with an attraction basin of at least unit Hamming distance plotted as a function of number of neurons.

has been carried out. In these simulations, we try to store as many as possible bipolar patterns in Hopfield nets with different number of neurons, such that all the patterns are made stable fixed points of the net, and further, each fixed point is surrounded by a *strict convergence ball* of radius 1. The results of both the Hebbian rule and our method are shown in Figure 3.15.

3.4.5 Conclusions

In this section we have presented a new approach for the design of discrete Hopfield net as an associative memory. Upon introducing the new concept of *activation difference* and relating it to the size of the *strict convergence ball*, we have developed certain techniques which lead to a Hopfield net with all the given patterns to be its stable fixed points, further, each fixed point is surrounded by a largest possible *strict convergence ball*. The result is compared favorably to those designed using the Hebbian rule.

Different from the Hebbian rule, our method results in an asymmetric non-zero diagonal weight matrix W. It follows that the conclusions about stability and the number of spurious states of a symmetric zero diagonal W are not applicable any more. From the formulation of the design task in the beginning of the chapter, it is obvious that what we care about are those states or vertices which fall in one of the *strict convergence balls* surrounding each stored pattern. We have also shown in the above presentations that if only the initial state of the net is inside some *strict convergence ball*, it will converge to the corresponding fixed point. So in this sense, the stability of the Hopfield nets designed using our method is guaranteed. As for the number of the spurious states, it might be the case that since we are using asymmetric weight matrix with non zero diagonal elements, this number could be larger than that of a symmetric zero diagonal weight matrix. But as discussed above, as long as the initial state of the net comes inside certain strict convergence ball, it will never converge to a spurious state but to a stable fixed point. For example, in the examples we discussed earlier there are always some spurious states outside the strict convergence balls. As presented in Subsection 3.4.3, an open problem of this approach is to give the theoretical proof to the algorithm which is used to find the optimal W so that $d_{\min}$ reaches its maximum.

3.5 CNN and associative memories

3.5.1 Introduction

Cellular neural networks (CNN) are introduced in [14] as an alternative to conventional computers for image processing and pattern recognition. This neural net structure has two important features: Its continuous-time operation allows the computational speed to depend only on the time-constant of the underlying dynamical system; Its local connection structure facilitates the analog VLSI implementation.

Given the basic neuronal dynamics, the rectangular arrangement of the neurons and the size of the connecting neighbors, the fundamental design problem for cellular neural nets is to find an appropriate set of connection weights for a specific processing problem.

There exist essentially two different design approaches depending on the nature of the processing problems: spatial operation and fixed-point programming. The former approach considers the cellular net as a spatial operator. With appropriate choices of the connecting weights, the net can function as a differentiator or an integrator or even some complex operator. In fact, this design approach has been illustrated with a number of image processing examples in [15]. It is worth noting that the spatial operations are generally divided into three classes: the cooperative operation, the competitive operation and the mixed operation. To see the relevance, the Laplacian template used in one of the examples in [15] is actually a version of "on-center off-surround" connection scheme which is a typical competitive operation ([23][39]).

The latter approach considers the cellular net more as a memory than an operator. The design task in this case is finding the connection weights so that a given set of binary patterns $p^1, p^2, \cdots, p^m$ are the stable fixed points of the net with prescribed size of basins of attraction.

Cellular neural nets designed in this fashion are termed *cellular associative memory* in general, though they are not necessarily confined to this specific domain of application.

Observe that while both of the design approaches try to shape the convergent dynamics of the underlying nets, completely different interpretations are given to the stable fixed points in the two designs. For the spatial operation design, the effect of the target operation is expected to be the fixed point. Given

data are simply set to the initial conditions of the nets. As any well-defined operation should produce a unique result, the design amounts either to establishing globally convergent dynamics, or to reshaping the size of the attracting basin around the desired fixed-point so as to cover all the expected data. We note in this case that any additional stable fixed-point indicates a potential hazard to the target operation, and is treated as a spurious state.

The fixed-point programming design, on the other hand, directly sets the given data to be the stable fixed-points of the nets. Note that if a design can guarantee a prescribed size of attracting basin for each stable fixed-point, then the issues such as spurious states, complete stability are totally irrelevant. In this case, the capacity consideration becomes the most important design issue.

With the attention focused mainly on the spatial operator type of cellular neural nets ([101][14][15]), no attempt has apparently been made to understand the essential features of the cellular associative memory, and to provide detailed design procedures. This section represents one attempt to deal with these important issues.

3.5.2 Discrete-time cellular associative memory

To simplify the analysis, we will consider exclusively a discrete time version of the cellular neural nets composed of linear threshold neurons. As the local connection is the main characteristic of the cellular nets, our simplification still captures the essential features of the continuous nets.

The discrete dynamics of the nets are defined when we specify a precise state updating rule at each discrete time instant. Here we choose to use the so-called synchronous updating, i.e., all the neuronal states are updated at the same time. To summarize, the cellular neural nets considered here have the following features:

- it is a rectangular array of neurons connected in a nearest-neighbor fashion as is shown in Figure 3.16. Note that such a configuration is termed as the 1-neighborhood connection following the terminology in [14];

- each neuron is a linear threshold element. In our case,

$$x_{ij}(t+1) = \text{Sign}[\sum_k w_k^{ij} u_k^{ij}(t)] \tag{3.35}$$

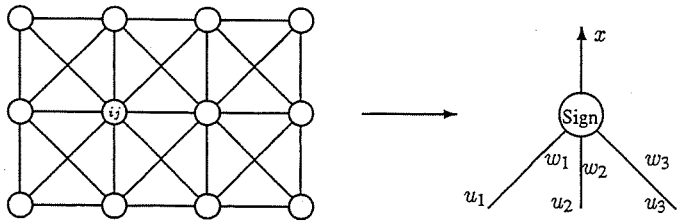


Figure 3.16: The configuration of the discrete cellular neural network.

holds at the time instance t. In (3.35), x_{ij} and w_k^{ij} are, respectively, the neuronal state and the kth connection weight for the ijth neuron. u_k^{ij}, being the kth input to the ijth neuron, is actually the previous neuronal state of its neighboring neurons, which makes (3.35) into a discrete dynamical system. All the neuronal states can be arranged into a matrix X with ijth entry denoting the ijth neuronal state. We call X the state matrix of the net;

- the state matrix X is synchronously updated, i.e., (3.35) is valid at any t and for any x_{ij};

- the net is symmetric. Thus every link is bidirectional, and there is only one weight for each link.

We note that the cellular net defined this way is actually a specification of the discrete Hopfield net with a synchronous updating rule [59]. However, with the essential restriction on the connection, the cellular nets will generally depict some specific properties rendering the results developed for the Hopfield nets useless in this context. Moreover, in considering the local connection pattern, we need to develop a completely different synthesis procedure. Hence, the relation to the Hopfield net is not pursued further.

3.5.3 Design rule and pattern conditions

Given a set of k binary (with entries of ± 1) $m \times n$ matrices $P^1, P^2, \cdots, P^k$ and a fixed retrieval range r (given in the sense of the Hamming distance, thus any given pattern differing from P^l by Hamming distance r, for some l, should retrieve P^l), we wish to store them in an $m \times n$ cellular net described in the previous section. Or equivalently, we wish to find a set of connection weights so that each of these binary matrices is made the fixed point of the net with the size of attracting basin r. Clearly, for such a storage to be possible, a design rule is needed which relates the connection weights to the pattern matrices to be stored. The simplest of such design rules is the Hebbian rule. When we denote by $w_{ij,pq}$ the weight connecting the ijth and pqth neuron, the Hebbian rule is simply

$$W_{ij,pq} = \begin{cases} 0 & \text{for } |i-p| > 1 \quad \text{or} \quad |j-q| > 1 \\ \sum_{l=1}^{k} P_{ij}^l P_{pq}^l & \text{for } |i-p| \leq 1 \quad \text{and} \quad |j-q| \leq 1 \end{cases}, \tag{3.36}$$

where P_{ij}^l is the ijth entry in the given pattern P^l.

Observe that the Hebbian rule (3.36) has two important features: It is local in the sense that any weight is determined by the states of the adjacent neurons; It is additive so that any additional patterns are easily incorporated into the existing design.

With the extent (m, n) of the net and the design rule essentially fixed, we expect certain limitation on k, the number of patterns that can be stored in the net. The maximum of such number is called the *capacity* of the net. Clearly, the capacity defined as such only indicates the best possible storage case. When the specific nature of the given sequences of patterns is taken into consideration, the actual number of patterns storable in the net is bounded by the basin requirement as well as the structure of the net. The subsequent development details the precise relations.

Here we follow a local analysis approach and try to generalize the result to the entire net. To proceed, let us observe that the typical connection pattern for each neuron (except those on the boundary of the net) is depicted in Figure 3.17. This can be treated as a basic 3×3 cell of the entire net. Let the operation of extracting a 3×3 cell centered around (i, j) out of a matrix X be represented

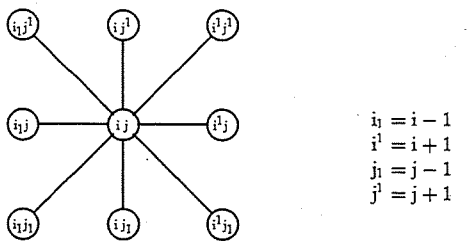


Figure 3.17: The typical connections for each neuron in a cellular neural net.

by the map N_{ij}:

$$N_{ij}(X) = \begin{bmatrix} x_{i-1,j+1} & x_{i,j+1} & x_{i+1,j+1} \\ x_{i-1,j} & x_{i,j} & x_{i+1,j} \\ x_{i-1,j-1} & x_{i,j-1} & x_{i+1,j-1} \end{bmatrix}. \tag{3.37}$$

Clearly, for a given sequence of stored patterns $P^1, P^2, \cdots, P^k$, the corresponding sub-sequence $S^1, S^2, \cdots, S^k$ can be defined by

$$S^l \triangleq N_{ij}(P^l) \quad l = 1, 2, \cdots, k.$$

Recall that all the weights connected to the ijth neuron can be determined by

$$W_{ij,pq} = \sum_{l=1}^{k} S^l_{pq} S^l_{ij} \tag{3.38}$$

for $i-1 \leq p \leq i+1, j-1 \leq q \leq j+1$. With this design rule and the sequence of sub-patterns $S^1, S^2, \cdots, S^k$ at hand, let us examine, for the 3×3 cell, the relations among the length of the sequence, the size of the attracting basin and the least sub-pattern distance. The result is summarized in the following theorem.

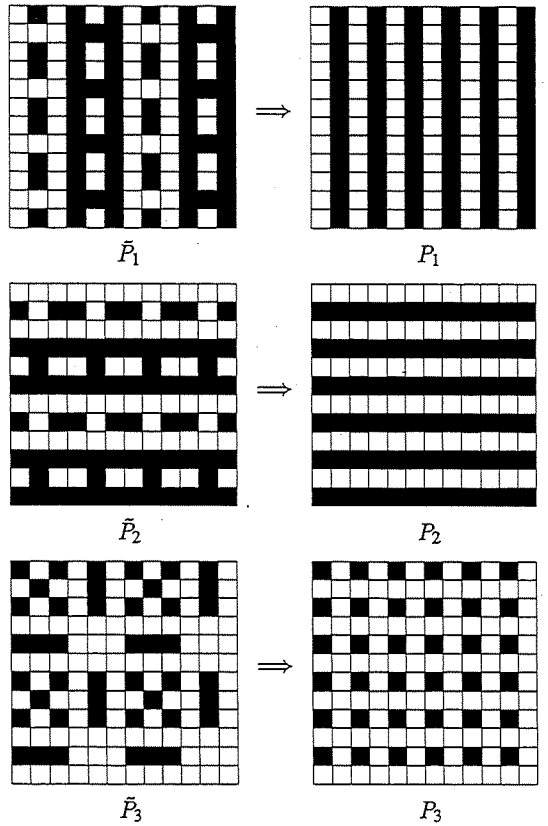


Figure 3.18: Pattern P_1, P_2 and P_3 have been stored in a CNN, when $\tilde{P}_i$ is presented as input, the CNN will output P_i $(i = 1, 2, 3)$.

Theorem 7 *Let h be the smallest Hamming distance among the sequence of the sub-patterns $S^1, S^2, \cdots, S^k$. Then a sufficient condition for the k sub-patterns to be stored with a retrieval range r using the Hebbian rule is given by*

$$9 - 2r - (k-1)(9 - 2h + 2r) > 0. \tag{3.39}$$

Proof:
First note that for the k sub-patterns corresponding to the 3×3 cell of the ijth neuron, we have

$$\sum_{pq} S^l_{pq} S^t_{pq} = \begin{cases} 9 - 2h & l \neq t \ \ (l,t = 1,2,\cdots,k) \\ 9 & l = t \ \ (l,t = 1,2,\cdots,k) \end{cases} \tag{3.40}$$

with $i-1 \leq p \leq i+1$, $j-1 \leq q \leq j+1$. Let a pattern $\tilde{S}^t$ be given within the Hamming distance r of the stored sub-pattern S^t (for some t, $1 \leq t \leq k$), then it is easy to see that

$$\sum_{pq} S^l_{pq} \tilde{S}^t_{pq} \begin{cases} \leq & 9 - 2h + 2r \quad l \neq t \ \ (l,t = 1,2,\cdots,k) \\ \geq & 9 - 2r \quad\quad\ \ l = t \ \ (l,t = 1,2,\cdots,k) \end{cases} \tag{3.41}$$

If $\tilde{S}^t$ is applied as an initial state, the input (without thresholding) to the ijth neuron can be written as:

$$U_{ij} = \sum_{pq} W_{ij,pq} \tilde{S}^t_{pq} = \sum_{pq} (\sum_{l=1}^{k} S^l_{pq} S^l_{ij}) \tilde{S}^t_{pq} = \sum_{l=1}^{k} S^l_{ij} (\sum_{pq} S^l_{pq} \tilde{S}^t_{pq})$$

In order for $\tilde{S}^t$ to converge to S^t, it suffices that the following two inequalities hold

$$(a) \quad S^1_{ij} = +1 \qquad U_{ij} \geq (9-2r)\cdot(+1) - (k-1)\cdot(9-2h+2r) > 0 \tag{3.42}$$

$$(b) \quad S^1_{ij} = -1 \qquad U_{ij} \leq (9-2r)\cdot(-1) + (k-1)\cdot(9-2h+2r) < 0 \tag{3.43}$$

We therefore arrive at (3.39). □

This local result for the 3×3-cell can easily be extended to the global net. Let the total number of the neurons in the entire cellular net be $M \times N$ with $M = 3m$, $N = 3n$, let $r_{\min}$, $r_{\max}$, $k_{\min}$ and $k_{\max}$ denote the minimum and the maximum of the obtained local retrieval range and the number of locally stored sub-patterns, respectively. Then it readily follows from Theorem 1 that C, the number of patterns which can be stored in the cellular net is bounded by

$$k_{\min}^{m \cdot n} \leq C \leq k_{\max}^{m \cdot n},$$

with the retrieval range d bounded by

$$m \cdot n \cdot r_{\min} \leq d \leq m \cdot n \cdot r_{\max}.$$

To have a concrete impression, we note that if the size of a cellular net is 30×30, and if the average number of the locally stored sub-patterns is 2 with the average local retrieval range 2, then $C = 2^{100}$ and $d = 200$, indicating a great potential of the cellular associative memories.

We also observe that the local analysis has revealed some fundamental restriction of the memory. Indeed, if we require that each 3×3-cell will store at least 2 sub-patterns, then (3.39) tells that we should have $h \geq 2r$. Thus even for a local retrieval range $r = 3$ (note that $0 \leq r \leq 9$) the minimal distance among the sub-patterns will have to be greater than 6 (note that $0 \leq h \leq 9$). This is obviously a serious limitation on the patterns to be stored. One should however bear in mind that Theorem 1 represents only a sufficient situation. In the actual design, the local relation among k, r h is far less restrictive than (3.39).

3.5.4 A simple example

A simple example is provided here to illustrate the actual design of a cellular associative memory. To this end, three patterns, P_1, P_2, P_3, in Figure 3.18 are to be stored in a 12×12 neuron cellular neural net by using the Hebbian rule (3.38). To retrieve these patterns, we apply the initial states $\tilde{P}_1, \tilde{P}_2, \tilde{P}_3$ shown in Figure 3.18 obtained by reversing 11% of the entries of P_i $(i = 1,2,3)$. Figure 3.18 indicates the desired memory performance.

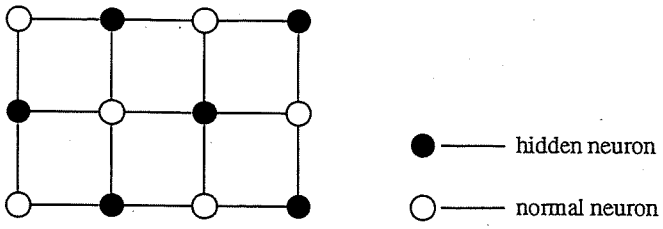


Figure 3.19: One possible modification of the standard cellular neural net

3.5.5 Conclusions

This section explores the possibility of using the cellular neural nets as associative memories. The Hebbian rule is employed to set up the connection weights. We have analyzed the memory storage features, and have provided, among other things, a sufficient relation among the number of stored patterns, the size of the attracting basins and the minimal pattern distance.

As our worst case study shows, the cellular associative memory may pose serious limitations on the kind of patterns to be stored. To overcome the limitation, we are currently investigating a so-called cellular net with hidden neurons. The basic idea of the structure is to introduce the hidden neurons in the original net structure in a uniform way. At each basic local cell, we can assign (see Figure 3.19) arbitrary states to the hidden neurons so as to alleviate the stringent requirement posed on the given sequence of patterns.

3.6 Conclusions and discussions

Since its being reintroduced in the late 1970s and early 1980s, artificial neural network has been one of the most active research areas in modern science. It has attracted the interests of researchers in many disciplines, ranging from electrical engineer, mathematician and physicist to psychologist and biologist.

Its known and potential applications cover almost all the subjects of electrical engineering and has extended to other areas like chemical engineering, ocean engineering, even economy and sociology. Although a systematic theory about artificial neural networks has yet to be established, many exciting results have been obtained (e.g., image processing by cellular neural networks presented in [14] and [15]) and some more powerful models are proposed (e.g., the ART family proposed by S. Grossberg and G.A. Carpenter in [24][12][13]).

Associative memory is one of the most promising potential application areas of artificial neural networks owing to the parallel computation nature and the convergent dynamical behaviour associated with some models of neural networks. Initiated by J.J. Hopfield in his two famous papers [42] and [43], a model of fully connected neural networks (called Hopfield net) has been extensively studied to function as an associative memory. Most of the previous work has focused on the properties and performance of the nets designed using the well known Hebbian rule ([59][17][102][65]) as well as the development of some other design rules (see [61][19] for some examples).

Under the so called *performance oriented* formulation of the design problem of neural associative memory, a data-driven approach has been proposed to design a memory with deterministic requirements on the fixed points or the size of attraction basin. First, by transferring the fixed points programming problems to solving a set of inequalities with the consideration of inconsistency, we use the feed forward nets and error back propagation algorithm to find the weights for the associative net. All the given patterns are made the fixed points of the obtained net with certain number of hidden neurons. Secondly, to meet the specification on the size of attraction basin, we pursue the model of Hopfield net with hidden structure even further. Through an orthogonalization process, the given patterns are made mutually orthogonal and the Hebbian rule is used for this set of orthogonal patterns to design an extended Hopfield net with hidden structure which guarantees all the given patterns to be its fixed points with the given size of attraction basin.

To overcome the problems of the inabilities of the Hebbian rule to guarantee any given patterns to be the fixed points with certain size of attraction basin, and at the same time, keep the number of the neurons used in the net minimum, a new method of designing Hopfield associative memory is developed as one of the important result of the thesis. Different from the Hebbian rule, this method

explicitly introduce the size of the attraction basin as a design object and ends up with a Hopfield net with all the given patterns to be the fixed points with the largest possible *strict convergence ball*. From the examples in Subsection 3.4.4, it can be observed that the size of this largest *strict convergence ball* is increased compared to those of the net obtained by the Hebbian rule.

As an important model of neural networks which can be easily implemented by VLSI, cellular neural networks are playing roles in several areas of applications (see [15],[88] and [82] for some examples). Most of the present applications of cellular neural networks are in the domains of the image processing and pattern recognition. To find its place for cellular neural networks in the area of associative memory, part of this thesis is devoted to the design of such a neural associative memory and the study of the capacity issue as well as the constraints on the patterns to be stored.

Based on the Hopfield net, a new design rule has been developed and some modifications to the standard Hopfield structure have been made. These efforts end up with a better performance of the neural associative memories compared to the traditional Hopfield associative memories, but the result is still not satisfactory. The main problem is: On the one hand, when we try to meet sharply all the requirements on the fixed points and the size of attraction basin, the net is probably stretched to an unreasonable size, i.e., with large number of hidden neurons; On the other hand, when we want to keep the number of the neurons relatively small, the size of the attraction basins will also be very small, especially when there are more patterns to be stored. It seems that we have to make our choice between the performance and the cost (in the sense of the number of neurons used) of the net. This is reasonable when the model is basically fixed (Hopfield net or its modified version).

It is now clear that to obtain the optimal performance-cost ratio of a neural associative memory, Hopfield model is not a good (or at least not the best) choice. A simple observation is that any neural network model of associative memories is not necessarily a dynamical one, because the retrieval process of the stored patterns in an associative memory can also be treated as a static mapping process. This and related issues will be discussed in detail in the following chapter.

Chapter 4

Pattern Association and Feedforward Model of Associative Memories

The objective of this chapter is to present a design technique that constructs a two-layer feedforward network for the realization of an arbitrary set of bipolar associations (p_i, q_i), $i = 1, \ldots, k$. The underlying idea of the technique is to use a layer of the hard-logic neurons to identify each p_i in the *winner-take-all* [107] fashion. Then the second layer of the so-called sign-neurons picks up the corresponding pattern q_i. An important feature of the two-layer net is that it can be used as an error-correcting associative memory if the thresholds of the hard-logic neurons in the first layer are properly adjusted. Another advantage of the technique is that neither extensive computation nor heuristics is involved in the determination of the network size, various weights and threshold values, making it possible to build feedforward nets for association problems of extremely large size.

4.1 Introduction

The design problem that will be treated in this chapter is the following: Let an arbitrary but well-defined[1] set of associations be given by $(p_i \rightarrow q_i)$, $i = 1, \ldots, k$, where p_i and q_i are bipolar vectors of dimension n and m, respectively. We want to find a layered feedforward neural network with linear threshold neurons that implements this set of associations. In other words, the net should be built such that when p_i is inserted as its input, its output will be q_i.

Sometimes, the input vectors p_i $(i = 1, \ldots, k)$ are subject to distortion, thus may differ in several bits from the original p_i. We want the net to be so designed that when a distorted input $\tilde{p}_i$ (different from p_i only up to a prescribed number of bits) is applied as the input to the net, q_i will still be generated as the corresponding output. It can be expected that this extra requirement may lead to some complication of the design.

Without explicit reference to the layered feedforward network structure, this abstract design problem has its origin in many areas including associative memories, cognitive inference systems, pattern recognition, logic circuit, error-correction coding, control, etc., and has been considered quite extensively in the past decades [105] [81] [84] [42] [52] [51] [26] [78] [83] [6] [41] [62] [72]. The existing design techniques and the resulting network structure generally fall into two classes depending on the underlying principles used. The first class of techniques utilizes the concept of dynamical convergence as the main principle for the associations. A neural network with feedback connections is needed in this case, so is an appropriate state updating rule. The Hopfield networks and the cellular neural networks are two examples in this class.

The second class of techniques relies on the feedforward structure of perceptron neural nets (see [105] for an interesting survey). Depending on how the weights and the thresholds are set, these design techniques can further be classified into *iterative design techniques* and *direct design techniques*. We shall discuss the idea and the limitation of these two types of techniques.

The essential idea of the iterative design techniques is *error correction*. A network structure (in terms of the numbers of layers and neurons in each layer) is set up *a priori*. An iterative algorithm is used to adjust the weights so as to reduce certain an error function for each presentation of p_i as the input of the

[1] The well-definedness is in the sense that $p_i \neq p_j$ for $i \neq j$.

net.

There are two widely-used forms of error functions, hence two different schemes for updating the weights (see [105]). The first one, called *single-step error correction*, modifies the weights to reduce the output error for the current input, say p_i. The famous *perceptron convergence procedure* proposed in [81] is based on this idea. Note that the single-step error correction, such as the perceptron convergence procedure, is effective only for single-layer feedforward nets. It is extremely difficult to figure out the errors for the hidden neurons based on the global output error, thus the updating of the weights for the hidden neurons can not be carried out.

The second scheme, called *the gradient method*, tries to reduce the mean square error accumulated for *all* the inputs to the net. If the mean square error is visualized as an error surface over a high dimensional space, the gradient method can be seen as moving along the error surface and trying to locate the global minimum. The problem with the gradient search is that it needs at least a partial understanding of the error surface on which the gradient descent is carried out. For layered feedforward networks of nonlinear neurons, even this moderate requirement is beyond our reach. We can, therefore, imagine that although the gradient method can be applied to multi-layer feedforward nets, there is no guarantee that a network will function exactly as required.

In contrast to the iterative design, the direct design techniques construct feedforward nets directly from the given set of associations $[p_i, q_i]$ $(i = 1, \ldots, k)$. (see, *e.g.*, [26][106][78][83][6][41]). Unfortunately, almost all the techniques in this class are designed to handle the dichotomy problem, *i.e.*, the dimension of q_i $(i = 1, \ldots, k)$ is 1. A common deficiency of the direct design is that the time spent in exploring the geometrical or algebraic structure of the given vectors and the associations grows exponentially with the dimension of the problem. This confines the applicability of the techniques to problems of small dimension, and/or limited number of associations.

The objective of this chapter is to present a new design technique for the implementation of an arbitrary set of bipolar vector associations by multi-layer feedforward nets. Compared to the afore-mentioned conventional techniques, our technique has the important features of being systematic, general and efficient. The basic network structure generated by the new technique is a generic two-layer feedforward net. There are two different kinds of neurons used for

the two different layers. Both the number of neurons in each layer and the corresponding weights are determined directly from the given set of associations without computation. When there is the error-correction requirement, the thresholds for the neurons in the first layer will be adjusted properly. Such an adjustment is straightforward and results in different error-correction ranges for the input vectors p_i $(i = 1, \ldots, k)$.

The outline of the chapter is as follows. In Section 4.2, the discrete neurons and the two-layer feedforward net to be used in our development are discussed. This discussion also serves to establish the nomenclature and conventions for the subsequent exposition. Section 4.3 presents the detailed construction of the network for a given set of vector associations. We also compare the new construction with the existing ones. Section 4.4 shows the way to modify the thresholds to meet the additional requirement of error-correction in the input vectors. A few illustrative examples are shown in Section 4.5. Additional remarks and final conclusion are contained in Section 4.6.

4.2 Neurons, multi-layer feedforward networks and vector associations

The feedforward neural net considered in this chapter has two layers as shown in Figure 4.1. Each of the layers consists of a set of computing elements called neurons. As this is a feedforward structure, the connection is only defined for neurons in adjacent layers. As the outputs of the neurons in the first layer are not directly available from the point of view of network output, all the neurons in the first layer are called hidden neurons. And the first layer itself is referred to as the hidden layer of the net.

The neurons considered here are multi-input single-output nonlinear functional elements with the following transfer function

$$y = f(\sum_i w_i x_i + \sigma), \tag{4.1}$$

where y is the output of the neuron; x_i is the ith input of the neuron, and w_i is its corresponding weight; σ is a constant threshold value; and $f(\cdot)$ is some nonlinear function. We are interested in two types of nonlinear functions, thus

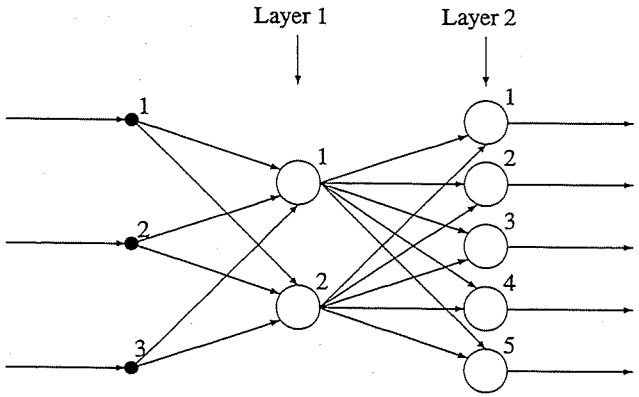


Figure 4.1: An example of the two-layer feedforward net. The output neurons are sign neurons and the hidden neurons are hard-logic neurons.

two types of neurons. The nonlinear function for the first type is given by

$$f_1(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0. \end{cases}$$

Such an $f_1(\cdot)$ is called a *hard logic* function. The corresponding neurons are naturally called *hard-logic neurons*.

The following nonlinear function $f_2(\cdot)$ defines the second type of neurons

$$f_2(z) = \begin{cases} 1, & z \geq 0 \\ -1, & z < 0. \end{cases}$$

This is the so-called sign function, thus the corresponding neurons are termed *sign neurons*. A conventional neural net is normally composed of a single type of neurons. As we shall see, it is in fact advantageous to use two different types of neurons in the same network. This is one of the important features of our approach to the design of the net.

Returning to the two-layer network structure, all the neurons in the first layer are the hard-logic neurons, and those in the second layer are the sign neurons. It will become clear later that this special arrangement will enable a simple implementation of the given associations.

For easy reference of the inputs and outputs of the nets, and of different neurons, weights and threshold values in the feedforward network, the following convention is followed. p_i, q_j are used to denote the ith input vector, respectively, the jth output vector of the two-layer feedforward nets. For a given two-layer feedforward net, we normally count its layers from the input to the output starting with the layer immediately following the inputs. To count the number of neurons in a particular layer, we start from the neuron at the top going downward. w_{ij} *in layer* k is normally used to refer to the weight on the link between the ith neuron in the $(k-1)$th layer to the jth neuron in the kth layer. Similarly, σ_i *in layer* k refers to the threshold for the ith neuron in layer k. We also speak of cross weight w_{ij} if $i \neq j$, and direct weight if $i = j$. Following the convention, it is relatively easy to refer to any particular neuron and its associated weights and thresholds. Figure 4.1 gives an example of our numbering convention.

4.3 Pattern association by two-layer feedforward nets

We now discuss the detailed design procedure to construct the feedforward net that realizes the given associations. To begin with, let a set of k *well-defined* associations be given by

$$p_i \rightarrow q_i \quad (i = 1, \ldots, k),$$

where p_i, q_i are bipolar vectors of dimension n, respectively m. As the first step of the construction, a two-layer feedforward net configuration is set up, which consists of the layer of the input nodes, the first and the second layer of neurons. The number of input nodes is n that is the same as the dimension of the given input patterns p_i. The neurons in the first layer, k in number (k is the number of the associations), are the hard-logic neurons. Finally, the total of m number of the sign-neurons will be allocated to the second layer. From the above arrangement, it is clear that the geometric size of our feedforward net is dictated by the length and the dimensions of the given patterns. It grows linearly with the size (the number of associations k) of the association problem.

Having fixed the configuration of the net, we can now assign the weight for each connection and the threshold for each neuron. The assignment is a straightforward substitution process, and no computation is necessary. More precisely, the weight connecting the ith input node to the jth neuron in the first layer is the ith entry of the pattern p_j $(i = 1, 2, \cdots, n, \quad j = 1, 2, \cdots, k)$; and the weight from the ith neuron in the first layer to the jth neuron in the second layer is the jth entry of the pattern q_i $(i = 1, 2, \cdots, k, \quad j = 1, 2, \cdots, m)$. In other words, the jth neuron in the first layer corresponds to the jth pair of matched patterns (p_j and q_j) and is connected to the input nodes by the corresponding elements of p_j and to the neurons in the second layer by the corresponding elements of q_j. Observe that this weight assignment process guarantees that all the weights will only assume either 1 or -1, which is a great advantage from the viewpoint of hardware implementation.

The thresholds of all the neurons in the second layer (*sign* neurons) are simply set to zero, whereas those neurons in the first layer (*hard-logic* neurons) are normally set in accordance with the error-correction requirement. When there is no such a requirement, we tend to set it to be θ, where $n - 2 < \theta < n$ and n is the dimension of the input patterns. In the rest of the section, we shall choose $\theta = n - 0.5$ for our analysis.

The design procedure of this special two layer feedforward network can be summarized in the following steps:

1. Construct a two layer network with n (the dimension of the given patterns p_i's) input nodes, k (the number of associations) hard-logic hidden neurons and m (the dimension of the given patterns q_i's) sign output neurons;

2. Choose the connecting weights from input nodes to the hidden layer to be the elements of p_i's and those from the hidden layer to the output layer to be the elements of q_i's;

3. Set the thresholds of the output neurons to be zero and those of the hidden neurons to be $n - 0.5$ (depending on the error-correction requirement).

Theorem 8 *The two-layer feedforward net designed in the preceding fashion will realizes the given set of associations*

$$p_i \rightarrow q_i \quad (i = 1, \ldots, k).$$

Proof:
Let us arrange the weights from the input nodes to Layer 1 into a $k \times n$ matrix W_1. The arrangement is such that the entries in the ith ($i = 1, \cdots, k$) row of W_1 are the weights from all the input nodes to the ith neuron of Layer 1. Further, the weights from Layer 1 to Layer 2 can be arranged into an $m \times k$ matrix W_2. In this case, the entries in the ith ($i = 1, 2, \cdots, k$) column of W_2 are the weights from the ith neuron of Layer 1 to all the neurons of Layer 2. Because of the ways that the weights are set, we obviously have the following equations

$$W_1 = \begin{bmatrix} {p_1}^T \\ {p_2}^T \\ \vdots \\ {p_k}^T \end{bmatrix}, \qquad W_2 = \begin{bmatrix} q_1 & q_2 & \cdots & q_k \end{bmatrix}, \tag{4.2}$$

where p_is and q_is are the given patterns (column vectors).

Similarly, if the threshold vector of Layer 1 is denoted by θ_1 (recall that the threshold vector of Layer 2 has been set to zero), we have

$$\theta_1 = \begin{bmatrix} n - 0.5 \\ n - 0.5 \\ \vdots \\ n - 0.5 \end{bmatrix}_{k \times 1}. \tag{4.3}$$

Upon denoting the n-dimensional input vector to the net by x, the k-dimensional output vector of Layer 1 by g and the m-dimensional output vector of the net by y, we can easily see that the function implemented by the feedforward net is the following

$$y = f_2(W_2 \cdot g) = f_2(W_2 \cdot f_1(W_1 \cdot x - \theta_1)), \tag{4.4}$$

where $f_1(\cdot)$ and $f_2(\cdot)$ are, respectively, the vectors of *hard logic* function and *sign* function defined previously.

Suppose the vector p_i ($i = 1, \ldots, k$) is applied as the input vector to the net, *i.e.*,

$$x = p_i, \tag{4.5}$$

then the output vector g of Layer 1 will be

$$g = f_1(W_1 \cdot p_i - \theta_1) = f_1\left(\begin{bmatrix} {p_1}^T \\ {p_2}^T \\ \vdots \\ {p_k}^T \end{bmatrix} \cdot p_i - \begin{bmatrix} n - 0.5 \\ n - 0.5 \\ \vdots \\ n - 0.5 \end{bmatrix}\right). \tag{4.6}$$

Since ${p_j}^T \cdot p_i < n - 0.5$ for $j \neq i$ ($j = 1, 2, \cdots, k$) and ${p_i}^T \cdot p_i = n > n - 0.5$, we have

$$g = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{4.7}$$

Or in other words, only the ith element of g is 1, and all the rest are 0.

Consequently, we have

$$y = f_2(W_2 \cdot g) = f_2(\begin{bmatrix} q_1 & q_2 & \cdots & q_k \end{bmatrix} \cdot g) = q_i, \tag{4.8}$$

which is what we have to prove. □

4.4 Association network as hetero-associative memory

We now show that the feedforward net designed in the preceding section can easily incorporate the error-correction feature. In this sense, the network functions as a hetero-associative memory.

Let us provide a short discussion on the hetero-associative memory before proceeding to the detailed design procedure. The hetero-association design problem is stated in its general form as follows: For a given set of bipolar associations $p_i \rightarrow q_i$ and a set of non-negative integers r_i $(i = 1, \ldots, k)$, we need to build a neural net such that when p_i is presented as the input of the net, q_i will be the output; moreover, all the input patterns with Hamming distance equal to or smaller than r_i to p_i should also give rise to the output q_i $(i = 1, \ldots, k)$. r_i is normally called the ith error-correction range. Because of the constraint imposed by the error-correction range, the input patterns $p_1, \ldots, p_k$ can not be completely chosen at will. In fact, the following condition must always holds for the input patterns

$$p_i^T p_j < n - 2r_i - 2r_j \tag{4.9}$$

for any $i \neq j$ $(i, j = 1, 2, \ldots, k)$ in order to avoid the overlapping among the adjacent error-correction ranges.

To visualize the condition (4.9), each p_i along with its error-correction range can be perceived as the center of the ith square with the side length r_i in the 2-dimensional plane. Thus, all the input patterns correspond to k such squares. (4.9) implies that these k squares will not overlap (see Figure 3.9 for a similar case). The arbitrariness of the input patterns is therefore confined to within the condition (4.9). Note that the condition (4.9) together with the well-definedness of the set of associations is the basis to judge if a given

hetero-association problem is meaningful. Any set of associations satisfying these two conditions are termed *proper*.

It is easy to verify that to incorporate the error-correction requirement on the feedforward net, the only modification needed is on the thresholds of the neurons in the first layer of the net, and it can be chosen as $n - 2r_i - 0.5$ for the ith neuron. Or in the following vector notation:

$$\theta_1 = \begin{bmatrix} n - 2r_1 - 0.5 \\ n - 2r_2 - 0.5 \\ \vdots \\ n - 2r_k - 0.5 \end{bmatrix}. \tag{4.10}$$

Suppose in this case, a vector $\tilde{p}_i$ instead of p_i $(i = 1, 2, \cdots, k)$ is presented as inputs to the net, and further, the Hamming distance between $\tilde{p}_i$ and p_i is not larger than r_i, which means that the corresponding pattern q_i of p_i should be the output of the net. With these notations, the Equation (4.5) becomes

$$x = \tilde{p}_i, \tag{4.11}$$

and the Equation (4.6) becomes

$$g = f_1(W_1 \cdot \tilde{p}_i - \theta_1) = f_1(\begin{bmatrix} {p_1}^T \\ {p_2}^T \\ \vdots \\ {p_k}^T \end{bmatrix} \cdot \tilde{p}_i - \begin{bmatrix} n - 2r_1 - 0.5 \\ n - 2r_2 - 0.5 \\ \vdots \\ n - 2r_k - 0.5 \end{bmatrix}). \tag{4.12}$$

Since ${p_j}^T \cdot \tilde{p}_i < n - 2r_j - 0.5$ for $j \neq i$ $(j = 1, 2, \cdots, k)$ and ${p_i}^T \cdot \tilde{p}_i = n > n - 2r_i - 0.5$, similarly we have

$$g = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{4.13}$$

Or in other words, only the ith element of g is 1, and all the rests are 0.

Consequently, we have

$$y = f_2(W_2 \cdot g) = f_2(\begin{bmatrix} q_1 & q_2 & \cdots & q_k \end{bmatrix} \cdot g) = q_i, \qquad (4.14)$$

which is what we need.

4.5 Design examples

Let us illustrate our design approach using two simple examples.

Example 1 (*Exclusive-Or Problem*). We first design a neural net for the famous *exclusive-or* problem:

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix} \rightarrow -1, \quad \begin{bmatrix} 1 \\ 1 \end{bmatrix} \rightarrow 1, \quad \begin{bmatrix} -1 \\ 1 \end{bmatrix} \rightarrow -1, \quad \begin{bmatrix} -1 \\ -1 \end{bmatrix} \rightarrow 1 \qquad (4.15)$$

Following the procedure discussed in Section 4.3, we can build a two-layer perceptron with 2 input nodes, 4 *hard-logic* neurons in Layer1 and 1 *sign* neuron in Layer 2. All the resulting weights and the thresholds are summarized in the following table:

The net configuration for the exclusive-or problem[2]

	w_{11}	w_{12}	w_{13}	w_{14}	w_{21}	w_{22}	w_{23}
Layer 1	1	1	-1	-1	-1	1	1
Layer 2	-1	*	*	*	1	*	*

	w_{24}	w_{31}	w_{41}	σ_1	σ_2	σ_3	σ_4
Layer 1	-1	*	*	1.5	1.5	1.5	1.5
Layer 2	*	-1	1	0	*	*	*

Note that while many conventional techniques can build a two-layer net of total 3 neurons for the present problem, our technique builds a 2-layer net of

[2]The symbol "*" is used to denote that the item is not defined.

total 5 neurons for the same problem. Thus, for trivial problems such as this, the new design technique does not really show its advantage except that the design is done in a much simpler way and the weights only assume two values.

Example 2. In this example, a feedforward net is designed for the associations of $(p_1 \rightarrow q_1), (p_2 \rightarrow q_2)$ and $(p_3 \rightarrow q_3)$. The patterns are given as

$$p_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad q_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad p_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \quad q_2 = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \quad p_3 = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ 1 \\ 1 \end{bmatrix} \quad q_3 = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} \qquad (4.16)$$

and the error-correction ranges are specified by $r_1 = 2$, $r_2 = 1$ $r_3 = 1$.

The input pattern condition (4.9) is obviously satisfied by the 3 given patterns as

$$p_1^T p_2 = -2 < n - 2r_1 - 2r_2 = 0,$$

$$p_1^T p_3 = -2 < n - 2r_1 - 2r_3 = 0,$$

and

$$p_2^T p_3 = -2 < n - 2r_1 - 2r_2 = 0.$$

where $n = 6$ is the dimension of the p_i's. Next, we construct a two-layer feedforward net with 6 input nodes, 3 neurons in Layer 1 and 3 neurons in Layer 2, and assign the weights and the thresholds shown in the following table

The net configuration for Example 2

	w_{11}	w_{12}	w_{13}	w_{21}	w_{22}	w_{23}	w_{31}	w_{32}	w_{33}	w_{41}
Layer 1	1	1	-1	1	1	-1	1	-1	-1	1
Layer 2	1	1	1	1	-1	-1	-1	-1	1	*

	w_{42}	w_{43}	w_{51}	w_{52}	w_{53}	w_{61}	w_{62}	w_{63}	σ_1	σ_2	σ_3
Layer 1	-1	-1	1	-1	1	1	-1	1	1.5	3.5	3.5
Layer 2	*	*	*	*	*	*	*	*	0	0	0

An important observation from both of the examples is that the design is carried out systematically with virtually no computation involved (only minor computations are needed in the determination of the thresholds in the first

layer). Another observation is that the resulting network is not necessarily greater than those generated by the conventional techniques.

4.6 Conclusions

We have presented a general design method that finds a two-layer feedforward neural net for an arbitrarily specified (but well-defined) bipolar association problem. We have also shown that the network can have the feature of error-correction if the thresholds for the neurons in the first layer of the net is properly set.

The obvious advantage in using a feedforward net for pattern association is that we only have to be concerned with algebraic rather than dynamical nature of the neural net. The latter is obviously much more intriguing and far less understood. It has been shown that with the feedforward net we have the perfect control over the size of the correction range. This is in contrast to the feedback net for which the correction range will have to be made as the basin of attraction of a stable fixed-point, which is nearly impossible to be controlled by changing the weights and/or the thresholds only.

In terms of VLSI implementation, the feedforward net that we have discussed above also has greater simplicity than the feedback nets. In this case, subtle issues such as stability need not to be concerned in the circuit design. The drawbacks seem to be that the net structure is problem dependent, and the two-layer structure does not allow modular implementation of the net. Thus when the underlying matching problem is changed or the dimensions of the given patterns go are increased, the circuit will have to be redrawn to reflect the change and the existing circuit is not usable. Another disadvantage seems to be that the thresholds of the first layer depends upon the dimension of the input patterns. This is somehow undesirable as the higher input dimension will lead to higher thresholds. However, this latter difficulty can be avoided by the rearrangement of the algorithmic order of the network computations.

Chapter 5

Rule-Based Neural Controller for the Inverted Pendulum System

In this chapter we will demonstrate how a heuristic neural control approach can be used to solve a complex nonlinear control problem. The control task is to swing up and stablize the cart-inverted pendulum system. Through the solution of this specific control problem, we try to illustrate a heuristic neural control approach with task decomposition, control rule extraction and neural net rule implementation as its basic elements. Specializing to the pendulum problem, the global control task is decomposed into sub-tasks, namely, pendulum positioning and cart positioning. Accordingly, three separate neural sub-controllers are designed to cater to the sub-tasks and their coordination, i.e., pendulum sub-controller (PSC), cart sub-controller (CSC) and the switching sub-controller (SSC). Each of the sub-controllers is designed based on the rules and guidelines obtained from the experiences of a human operator. A simulation result is included to show the actual performance of the controller.

5.1 Introduction

Many industrial control problems can simply be phrased as choosing right inputs to bring the state of a plant to some desired position in an appropriate state space. A typical example is *stabilization*, in which the desired control action is to bring the state of a plant to zero by designing appropriate control inputs. Another example is the shift of set-point. In this case, control inputs are sought in order to bring the state of a plant from a previous set-point to the intended one. Control problems as such can be extremely involved even for relatively simple nonlinear systems, no general methodology is available for actually carrying out the design.

There exist, however, certain classes of nonlinear control problems for which the allowable control inputs are restricted to a few fixed functional types. An extreme case is that the control can only assume two different constant values. Control problems of this kind can usually be translated into finite decision problems that can then be solved using techniques such as simple neural nets [104][91].

The idea of translating control problems into decision problems can also be extended beyond the classes mentioned above. In fact, if the state-space of a plant is partitioned into a finite number of regions, a continuous control action defined over the state-space can, under some circumstances, be approximated by a constant in each of the regions. If we further confine the input to have only a finite number of levels, then the control problem can readily be treated as a finite decision problem. Observe that this idea is also the principle behind the so-called fuzzy control approach [89], though the state-space partition in fuzzy control does not generate regions with crisp boundaries.

An obvious advantage of this decision approach to control problems is that the controller can be designed in a model-free fashion. This is in contrast with the traditional control design methodology where the model of a plant is essential to the control design. Originated by Widrow [103], the decision approach points to an interesting direction in handling the classes of difficult nonlinear control problems. Other pioneering works along the same line can be found in [21][71][86], though these works typically follow a stochastic rather than deterministic formulation.

The decision problems talked above are, after all, the problems of pattern

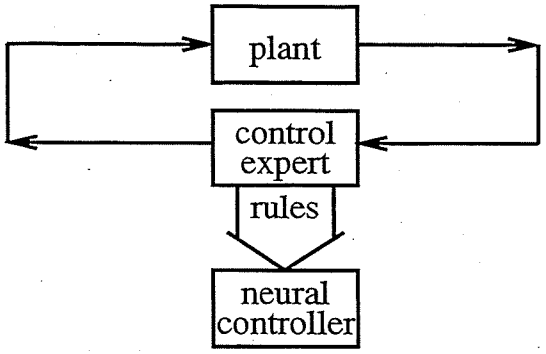


Figure 5.1: The scheme of the rule based controller design.

recognition. This equivalence follows from the fact that deciding which control input should be assigned to a region in the state-space is not different from deciding which class the region should belong to. As the number of inputs to be chosen from is finite, the pattern recognition problem is clearly well-defined. Unlike an ordinary pattern recognition problem, however, the decision boundaries in this case are generally determined by the underlying dynamics of the plant as well as the precision of the approximating system (if approximation is involved). One can imagine that decision boundaries can be extremely complex, and hard to determine even for simple nonlinear plants with detailed mathematical models.

The pattern recognition problem can be approached with many different conventional techniques. Noticeable for its relative versatility among many of these techniques is the so-called neural net approach. Neural nets can come in many different varieties, and a widely-appreciated family of neural nets is the layered feedforward net with bipolar neurons.[1] This structure is called *Madeline* by Widrow [104], and is the limit case of the well-known back-propagation net [84].

An important technique that can be used to design feedforward nets for

[1] Bipolar neurons are the neurons with the output of either $+1$ or -1. A neural net with bipolar neurons will simply be referred to as bipolar neural net.

the determination of decision boundaries using the feedforward neural nets is the so-called *supervised learning* proposed in [104]. The essence of this technique is to use the least mean square (LMS) algorithm (or its variations depending on the structure of the nets) to train the neural nets for selected input patterns and the corresponding targets. The trained neural nets will hopefully correctly generalize to the untrained patterns. The difficulty in applying the supervised learning for actual control problems is that it is not always clear what the targets are for the neural nets to learn. Besides, the uncertainty in the choice of pattern representation and in the generalizability of the nets turn out to be essential stumbling blocks for the technique to work. Another design technique is called *direct design* [78][26]. We shall see later that this technique is ideal for coding rules directly into a feedforward net.

Bipolar neural nets with feedback connections are also useful in solving the pattern recognition problem. A feedback neural net is able to carry out dynamical operations, thus suitable in dealing with time related decision problems. However, due to the complexity in the analysis and the design, the potential of the feedback neural nets has not been fully explored in the context of decision control. No systematic learning or direct design techniques are available. One design technique that we shall employ in this chapter is to directly code the extracted dynamical control rules into a feedback neural net in much the same way as we design a feedforward net.

The neural decision control approach we shall explore consists of three basic steps: Task decomposition, control rule extraction and neural net rule implementation. We start by decomposing a complex nonlinear control task into manageable sub-tasks that are in turn translated into decision problems. Note that decomposition is one of the principles widely used in AI. For each of the decision problems, the heuristic decision rules concerning individual problem and their coordination are extracted. The final step is to code the rules for individual decision problems and coordination into separate sub-neural nets. These neural nets can be feedforward or feedback bipolar nets, and their proper combination will give rise to the desired global neural controller. The idea of the approach is illustrated in Figure 5.1.

Apparently, this approach is heuristic in nature and is along the same line as the well-known fuzzy and expert control approaches that are built upon the set of heuristic rules. The distinct features of our approach are, perhaps, the idea

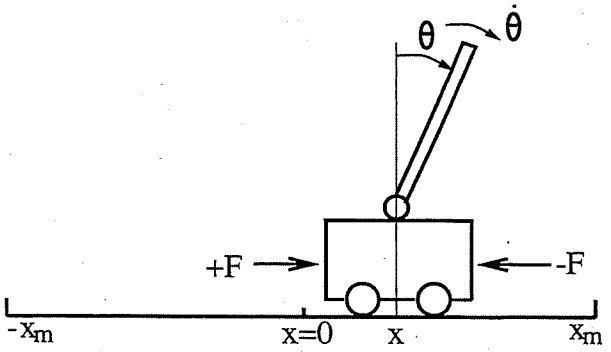


Figure 5.2: The cart-pendulum system, the angle and angle speed are positive in clockwise direction.

of dynamical rules, and the way the rules are realized. In fact, these ideas may lead to a substantial reduction in the number of rules required, consequently result in a compact feedback neural net controller. We shall highlight the differences in the concluding section of this chapter.

To demonstrate the relevance of the approach to real world problem solving, we shall apply it to this truly nonlinear control problem: Change the set-point of the well-known inverted cart-pendulum system. Figure 5.2 shows the setup of the cart-inverted pendulum system. Note that $\pm X_m$ is the pre-set limit of the track.

The control problem described above is a nontrivial problem of nonlinear regulation. It is apparently more difficult compared to the pendulum balancing problem (and its variations) widely adopted as a benchmarking test system for various neural controllers [104][5][100][25][2]. The latter control problem is linear in nature, thus does not really serve to illustrate the strength of the neural controllers that are supposed to be apt at handling hard nonlinear control problems. In fact, the balancing problem can be solved more elegantly by using techniques developed in linear control theory.

The rest of this chapter is structured as follows. In the subsequent section the overall control scheme of the system is presented and the control problem is decomposed into sub-problems and so is the controller. The design of each

of the sub-controllers and the switching mechanism between them is detailed in Section 5.3. Section 5.4 presents the simulation results of the control system followed by the conclusions and additional remarks in Section 5.5.

5.2 The Plant dynamics and problem decomposition

Though the precise mathematical model of the inverted pendulum system is not directly relevant to the design of our controller, we nonetheless write it down here to show the nonlinearity inherent in the system, and set the notations. With appropriate setting of the parameters, these equations are also used to generate the next state based on the current input and state of the system, which is the basis of our simulation and design.

The inverted pendulum system shown in Figure 5.2 consists of a cart of mass m and a pendulum of mass m_c centered at the half length l. The constant force F is applied in both horizontal directions through the mass center of the cart so as to move it up and down the track. The track is delimited at both ends as shown in Figure 1 by $\pm X_m$. According to [11], the system is modeled by the following set of differential equations

$$\ddot{\theta} = \frac{g \sin\theta + \cos\theta[\frac{-F - ml\dot{\theta}^2 \sin\theta}{m_c+m}]}{l[\frac{4}{3} - \frac{m\cos^2\theta}{m_c+m}]} \tag{5.1}$$

$$\ddot{x} = \frac{F + ml[\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta]}{m_c + m}, \tag{5.2}$$

where θ (radian) and x (meter) are respectively, the angular and the cart displacements; $g = 9.8m/s^2$ is the acceleration due to gravity. To be more specific, the parameters are chosen as $F = 10N$, $m_c = 1.0kg$, $m = 0.1kg$, and $l = 0.5m$ for the purpose of controller design. Note that the friction between the cart and the track, and the one at the hinge between the pendulum and the cart are all neglected in the above equations to make them look less formidable.

Obviously, even start with the known Equations (5.1) and (5.2), the control does not seem to follow from any existing linear techniques as the control problem itself will not allow the linearization of the mathematical model. Moreover, there seems to be no nonlinear control technique that solves the

problem in a systematic manner. It is obvious that a complex nonlinear model even to the extreme detail may, in a lot of cases, still be unable to help to determining the appropriate control[2]. This is where a model-free control approach (heuristic approach) in a sub-class is preferred.

To motivate our model-free heuristic design approach, let us first analyze how a human operator would try to accomplish the desired control objective. Naturally, the first step towards the control involves a trial-and-error process of getting a feel of the system. This is a process of rule extraction. The extracted rules are situation-action in nature and completely different in nature from the pure mathematical modeling of the dynamics of the system.

After getting a feel of the system, the next step the human operator is likely to take is to identify that the pendulum swing-up operation and the cart-positioning can actually be viewed as two separable sub-control objectives. To meet the final control demand, he only has to first swing-up the pendulum regardless of the cart disposition (within the pre-set limits), then repositioning the cart back to the origin of the track while balancing the pendulum. The later two sub-problems are apparently considerably manageable compared to the original one.

In fact, by playing with a computer emulation program of the inverted pendulum system, the author of this thesis has gone through exactly the same steps described above to reach the control objective. It follows from his own experiences that trying to simultaneously take care of swinging-up and the repositioning proves to be extremely difficult. This can perhaps be interpreted in terms of trajectory flow in the state-space composed of states $\theta, \dot{\theta}, x, \dot{x}$. The feasible trajectory may first wander far away from the origin before it eventually comes back. Though the decomposability of a complex problem seems generically possible, the actual process of decomposition is not as simple at all as shown in this example. It may involve analysis, experiences through trial and error, intuition and generalization.

The actual process of decomposition and rule extraction for a particular system is influenced by the nature of the underlying plant as well as the control requirements. For instance, the decomposition and rules will be different if an

[2] In this chapter, the model refers, specifically, to the mathematical equations of the systems. Thus, a set of heuristic rules describing the behaviour of a system is not considered as the system model.

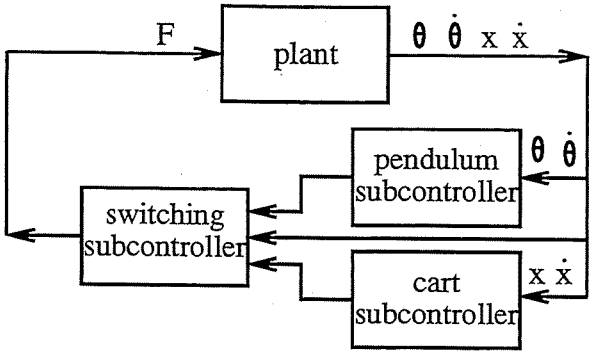


Figure 5.3: The overall control scheme.

additional requirement of time-optimality is required.

Although the task of decomposition is often explicitly realized and attempted by a human operator, the aspects of rules he comes up to deal with each sub-problem and coordination of the rules for different sub-problems are less explicit. For example, while playing with the emulation program, we can not substantiate our instant decisions that eventually contribute to an eventual success. This tacitness of the rules is often the most difficult aspect of all rule-based approaches. The degree of difficulty in obtaining rules varies for different control problems. In the following section, we shall work out the appropriate rules for the inverted pendulum system to hint the general principles we normally follow.

5.3 Rule extraction and neural net implementation

We now discuss how the control rules are extracted and implemented by simple feedforward and feedback neural nets.

As discussed before, the two sub-problems, namely, swinging-up the pendulum and repositioning the cart to the origin, will be separately dealt with using two sub-controllers, Pendulum Sub-Controller (PSC) and Cart Sub-Controller (CSC). The actions of the two sub-controllers are coordinated by

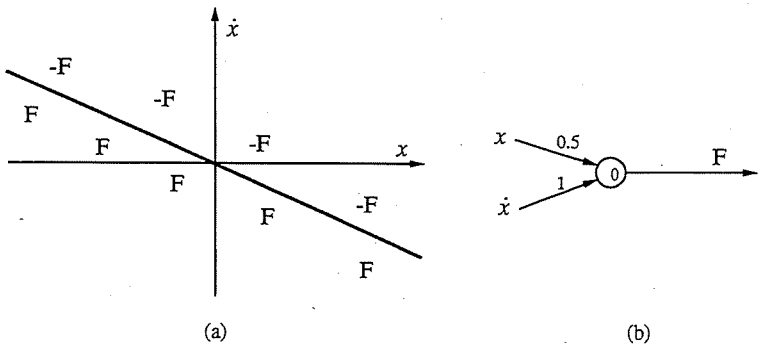


Figure 5.4: Geometric implementation of neural network. (a) Rule in geometric form. (b) Neural network implementation.

the third sub-controller called Switching Sub-Controller (SSC). The overall decomposed control scheme can be shown in Figure 5.3. The rule extraction processes and neural net rule implementation for all the three sub-controllers are detailed in the subsequent subsections.

5.3.1 Neural net implementation

As presented earlier, the weights and thresholds of a neural network can be implemented either through learning or using direct method. In this work we use a geometric method to implement rules in neural network. To illustrate how geometric method works, we will give a simple example. Suppose we have a simple rule such as:

$+F$ **when** $0.5x + \dot{x} > 0$, **otherwise,** $-F$.

This rule can be expressed graphically in the state space of x and $\dot{x}$ as shown in Figure 5.4(a), while Figure 5.4(b) shows the neural implementation of this rule.

5.3.2 Pendulum sub-controller

The sub-controller for swinging-up and stabilizing the pendulum is viewed as consisting of three functional parts: Pendulum swinging-up, stabilization and switching from swinging-up action to stabilization.

The switching condition between the two control actions is a prescribed small angular displacement α ($\alpha > 0$), i.e., whenever $-\alpha < \theta < \alpha$, the switching will take place.

Experience shows that it is impossible to swing up the pendulum to the range of $-\alpha < \theta < \alpha$ by applying a constant force along a single direction (it may become possible if the magnitude of the force is extremely large). We will have to switch the direction of the force at a certain point (say, $\dot{\theta} = \mu$) of the state space. If the specified magnitude of F is small, more than one such force switching is needed, but if this magnitude is too small, it is impossible to swing up the pendulum no matter how many times the forces are switched. For our case ($F = 10$), one switching is enough if μ is suitably chosen.

The behavior of the system under such control input sequence is as follows. When $+F$ is applied, the pendulum (starting from the initial position of $\theta = \pi$, $\dot{\theta} = 0$) swings up to a larger θ on the side of $\theta > \pi$ with $\dot{\theta} > 0$, until $\dot{\theta}$ decreases to zero (at which point $\theta - 2\pi < -\alpha$). Then the pendulum begins to fall down with a negative $\dot{\theta}$. When $\dot{\theta}$ becomes equal to μ, the force changes to $-F$ resulting in the pendulum falling down with an increasing negative $\dot{\theta}$. When the pendulum swings back to the point $\theta = \pi$, there will still be a negative $\dot{\theta}$ (note that without changing the direction of F from $+F$ to $-F$ at $\dot{\theta} = \mu$, $\dot{\theta}$ will be zero at this point). By continuously applying $-F$, the pendulum will swing up to the point where $\theta < \alpha$ on the side of $\theta < \pi$.

Following the preceding analysis, it can be readily seen that the choices of α and μ are vital to the success of this sub-controller. The improper choices of α and μ will result in either being unable to swing the pendulum up with a single change of forces or the oscillation between the swing-up and the stabilization actions. Several trial-and-error processes indicate that one of the proper choices is $\alpha = 0.2$ and $\mu = -1$.

It is relatively simple for the design of the stabilizing part of PSC. Based on our intuition, it is easy to see that we should apply $+F$ when $\theta > 0$ and $\dot{\theta} > 0$, apply $-F$ when $\theta < 0$ and $\dot{\theta} < 0$, and finally, apply $+F$ or $-F$ when $\theta \cdot \dot{\theta} < 0$ depending on the ratio $\kappa = \dot{\theta}/\theta$ (κ will be chosen to be -15).

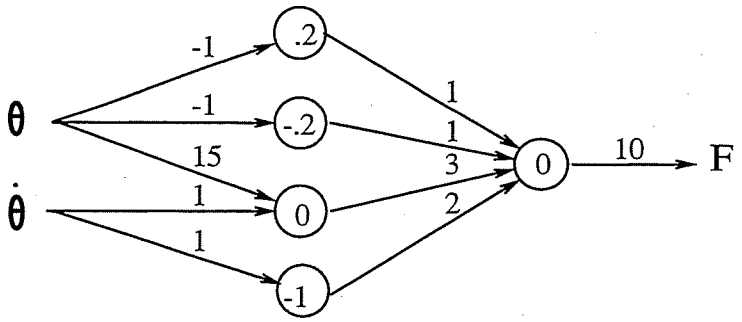


Figure 5.5: The Pendulum Sub-Controller.

To summarize, the rules for the design of PSC are given below:

Rules for PSC:

- $+F$ when $\dot{\theta} > \mu$, otherwise, $-F$ (swinging up);
- $+F$ when $\kappa\theta + \dot{\theta} > 0$, otherwise, $-F$ (stabilization);
- change from swing-up to stabilization when $-\alpha < \theta < \alpha$.

From the geometric theory of neural networks [78][26], we know that these rules can easily be implemented in a two layer multilayer perceptron neural network. The details are shown in Figure 5.5. In the figure, the values of the weights are placed close to their corresponding links, and the values in the circles indicate the thresholds of the corresponding neurons. Note that both the weights and thresholds of the neural net are either the parameters discussed above (α, μ, κ) or those chosen for the purpose of logic operations.

5.3.3 Cart sub-controller

The position control of the cart is quite simple. The direction of the applied force is determined by x and $\dot{x}$, the position and the speed of the cart. When $x > 0$ and $\dot{x} > 0$, $+F$ should be applied; when $x < 0$ and $\dot{x} < 0$, $-F$ should

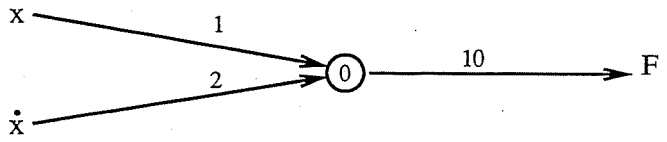


Figure 5.6: The Cart Sub-Controller.

be applied; when $x \cdot \dot{x} < 0$, the choice of $+F$ or $-F$ depends on the ratio $\rho = \dot{x}/x$. Based on the experience, ρ has been chosen to be -0.5. So the rules for CSC are the following.

Rule for CSC:

- $+F$ when $\rho x + \dot{x} > 0$, otherwise, $-F$.

The neural network implementation of this sub-controller is shown in Figure 5.6.

5.3.4 Switching sub-controller

Although the two sub-controllers (PSC and CSC) designed above are able to carry out separate pendulum swing-up and cart positioning control actions, they need to be combined dynamically by the switching sub-controller (SSC) to finally realize the global control objective. The idea of switching between the two sub-controllers is based on the following simple fact: When the pendulum is within a very small range of zero state ($-\beta < \theta < \beta, -\nu < \dot{\theta} < \nu$, where β, ν are small positive values), it will need more reverse forces to bring the pendulum back to this range than to push it away. This phenomenon can be roughly explained by the torque applied on the pendulum, i.e., the smaller $|\theta|$ is, the larger this torque will be when F remains the same.

Based on the above phenomenon, we will design the SSC to work in the following way: Whenever $-\beta < \theta < \beta$ and $-\nu < \dot{\theta} < \nu$, switch the control to the CSC which will generate a sequence of identical forces ($-F$ or $+F$ depending on the state of the cart); the actions of CSC will cause the pendulum

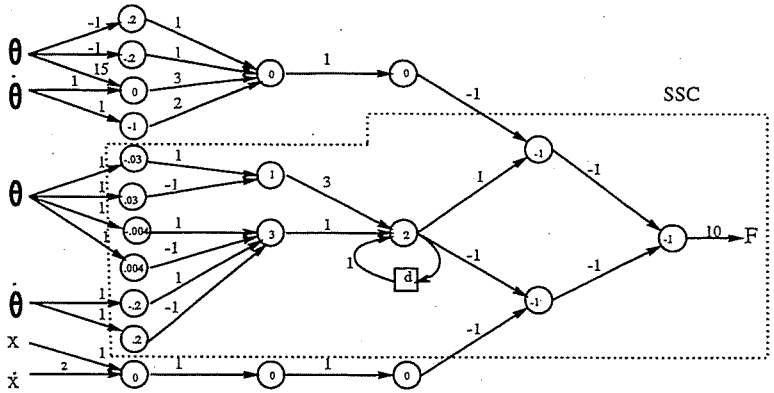


Figure 5.7: The whole neural controller with SSC being highlighted by dotted lines. The symbol [d] represents a unit time delay.

to leave the small region around zero state till the point that $\theta > \gamma$ or $\theta < -\gamma$ (γ is a small angle but greater than β), the SSC will switch the control to the PSC which will bring the pendulum back to the zero state and also move the cart toward the origin. This switching process will continue until the cart reaches the origin of the track, after which both the pendulum and the cart will swing back and forth within a small range of zero state. Let us summarize the rules for SSC as follows.

Rules for SSC:

- switch to CSC when $-\beta < \theta < \beta$ and $-\nu < \dot{\theta} < \nu$;
- switch back to PSC when $\theta < -\gamma$ or $\theta < -\gamma$.

Because of the dynamical nature of this switching mechanism, we have used time-delay MLP (a kind of simple feedback neural nets) for the implementation of SSC. Together with PSC and CSC, the SSC is shown in Figure 5.7 as part of the whole neural controller. In this realization the parameters have been chosen as follows, $\beta = 0.004$, $\nu = 0.2$ and $\gamma = 0.03$.

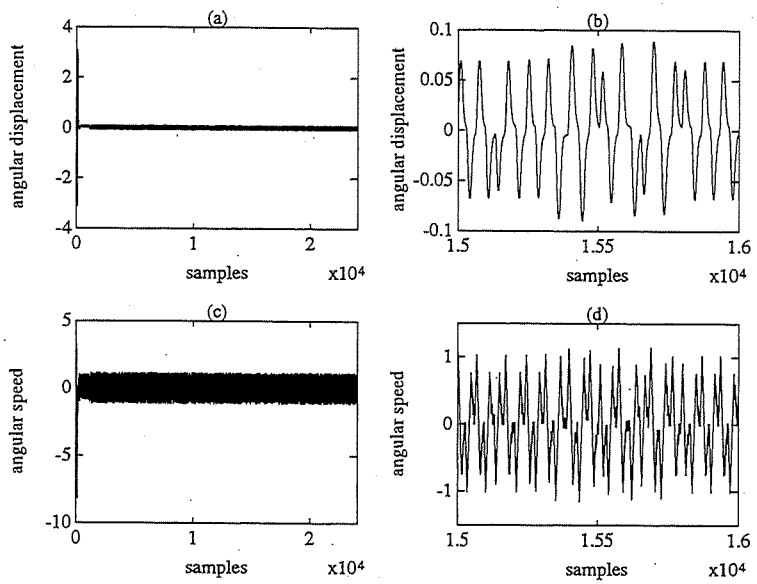


Figure 5.8: The result of simulation (pendulum). (a) shows the whole evolution process of angular displacement against the time samples. (b) is a zoom-in picture of part of (a) from which one can clearly see that the pendulum moves closely around the zero point irregularly after stabilization. (c) shows the whole simulation process of angular speed against the time samples. (d) is a zoom-in picture of part of (c) which indicates the pendulum keeps swinging back and forth after stabilization.

5.4 Simulation

The simulation of the neural controller for the inverted pendulum system is done on a workstation using the software package Matlab. The sampling period is 0.01s and a typical simulation lasts for about 24000 samples, which is equivalent to 4 minutes in real time. One of the simulation results is shown in Figure 5.8, Figure 5.9 and Figure 5.10.

In Figure 5.8(a), the angular displacement of the pendulum is shown over the whole simulation process, while 5.8(b) shows the "zoom-in picture" of the samples from 15000 to 16000. Figure 5.8(c) and 5.8(d) are the counterparts for the angular speed of the pendulum. It can be seen from these figures that after a very short swing up period, both the angular displacement and the speed of the pendulum fall within a small range of zero and oscillate irregularly within that small range. In order to see clearly the behaviour of the pendulum during its swing up process, we show in Figure 5.9 the values of angular displacement and angular speed versus time samples for the first 500 samples.

Figure 5.10 demonstrates the behaviour of the cart. Similarly, 5.10(a) and 5.10(c) show the whole evolution processes of the position and the speed of the cart respectively, 5.10(b) and 5.10(d) give detailed descriptions of the two processes for samples from 15000 to 16000. It is easy to see that the stabilization of the cart is achieved after the stabilization of the pendulum and the cart is moving within a small range around the origin of the track after a short period of time.

5.5 Discussions and conclusions

We have presented a heuristic neural control approach for solving nonlinear control problems. This approach is decision based, and has task decomposition, rule extraction and neural net rule implementation as its essential elements. Via the specific control problem of the inverted pendulum, we have demonstrated how each of the three steps is actually considered and realized.

It may be of interest to compare this approach to other rule-based control approaches such as fuzzy control and expert control. It turns out that the rule-based neural control has the following important advantages. First, it is able to implement dynamical decisions or rules rather than just the static mapping

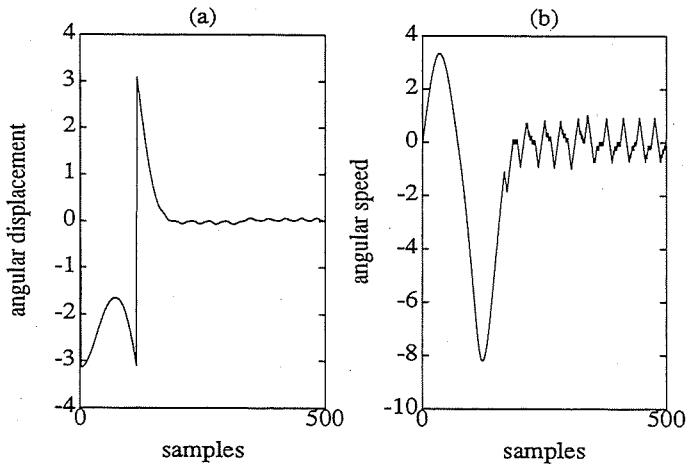


Figure 5.9: The first 500 samples of simulation (pendulum). The discontinuity of the curve in (a) is because that we have chosen the range of the angular displacements to be $[-\pi, \pi]$ and these two values should be looked upon as the same position for the pendulum.

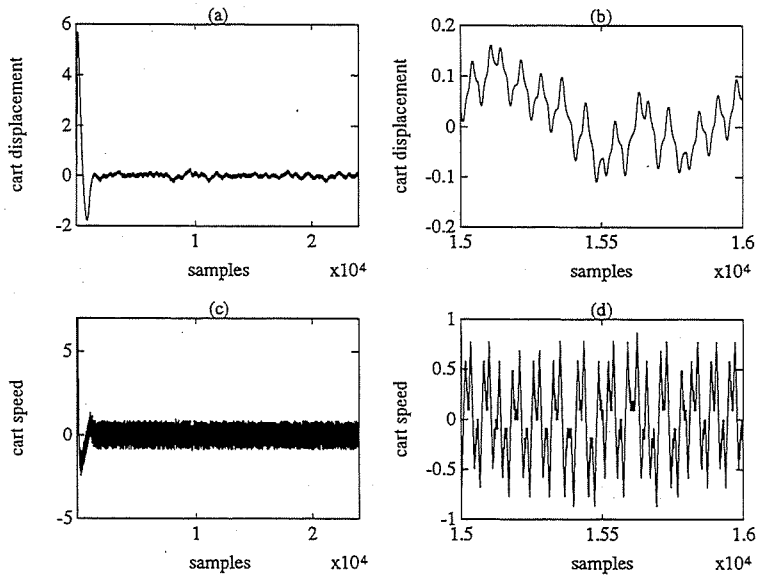


Figure 5.10: The result of simulation (cart). (a) shows the displacement of cart against time samples for the whole simulation process. (b) is a zoom-in picture of part of (a) in which it is clearly shown that the cart moves irregularly within a small range of the origin of the track. (c) shows the speed of cart evolves against time samples. (d) shows a detailed picture of (c) from which one can see that the cart keeps moving towards or away from the origin of the track.

actions. More precisely, in our feedback neural controller, the control action based on the outcome of a sub-decision process or rule can be dependent on not only the current state of the system, but also on the previous outcome of its own. This dynamical decision process, lacking in both fuzzy and expert control approaches, strengthens considerably the decision ability, and sometimes proves to be indispensable for the decision control problems. In fact, it is the problem decomposition that necessitates the dynamical decision process. And decomposition is often considered to be a key to complex control problems.

As a fringe benefit, dynamic decision process requires fewer and simpler rules than a pure static decision. This is easily understood as in the static case, time completely unfolds resulting in the need to cover all the time aspects of the decision.

Secondly, once constructed, the rule-based neural controller is a deterministic nonlinear dynamical system (with local feedback only), thus allowing the analysis to be carried out for performance evaluation. This is in contrast to both fuzzy and expert controls, where their final linguistic constructs prevent an analytic evaluation.

Finally, because of its parallel nature the rule-based neural controller is computationally advantageous over fuzzy and expert control. The computational advantage also follows from the reduced number of rules as a result of dynamical decision process.

Like all other rule-based approaches, the rule-based neural controller is inherently non-analytic and imprecise. The ambiguity follows from the heuristic and subjective nature of the problem decomposition and rule extraction. Moreover, the approach does not intend to solve all complex nonlinear control problems. What it really attempts to solve is the nonlinear control problems that are less well-defined, but nevertheless seem intuitively solvable.

Chapter 6

Predictive Control of Nonlinear Systems by Feedforward Networks

In this chapter, by using the properties of universal approximation of multilayer feedforward neural network, a class of discrete nonlinear dynamical systems are modeled by a network of two hidden layers. The backpropagation algorithm is then used to train the model to identify the nonlinear systems to a desired level of accuracy. Based on the identified model, a one-step-ahead predictive control scheme is proposed in which the future control inputs are obtained through some nonlinear optimization process. Making use of the on-line learning properties of neural networks, the predictive control scheme is further developed into an adaptive one which is robust to the incompleteness of identification. Simulation results show this neural control scheme works well even for some very complicated nonlinear systems.

6.1 Introduction

Adaptive control, together with robust control and optimal control has been one of the main active research topics in the area of system identification and control for the past decades. Significant progresses have been made in

the adaptive identification and control of time-invariant linear systems with unknown parameters [3][67]. The methods used to choose the model of identification and control are based on the well developed linear system theories and the stable adaptive laws for adjusting the parameters of the controller and identifier were established. In recent years, efforts have also been made to develop adaptive identification and control schemes for nonlinear plant models [99][87] which is partially due to the advances in nonlinear control theory and, in particular, feedback linearization techniques [48][70]. However, this area which is known as *adaptive nonlinear control*, deals mainly with those nonlinear systems which have unknown parameters appearing linearly with respect to known nonlinearities. As a result, adaptive nonlinear identification and control schemes for the more general nonlinear dynamical systems are yet to be developed. The work reported in this chapter represents an effort of using neural networks to this end.

Neural networks have been successfully used in the areas of speech recognition, image processing, optimization and classification, etc. Some efforts have also been made to use neural network models to deal with some specific nonlinear control problems [69][5][33][50][77]. The rule based neural controller for the inverted pendulum system discussed in Chapter 5 represents a typical examples of these methods. But the specific nature of these methods prevent them from being generalized to the broader category of nonlinear control problems.

Recently a couple of results are reported on the use of neural network models for identification and control of general nonlinear systems. Among them, the most significant contribution was made by Narendra and Parthasarathy in [68]. In their paper, they propose a systematic method to identify and control a general nonlinear plant by multilayer feedforward neural networks. The models they used for control and identification are mostly derived from the results established in the traditional adaptive control theory, e.g., parallel and series-parallel identification, algebraic and analytic parts of adaptive control problems, direct and indirect control, etc. The algorithms they used to train the networks are the standard back-propagation and the so-called "dynamical back-propagation".

Another important contribution to the topic of neural control and identification was made by Polycarpou and Ioannou in [76]. In this report, a class

of frequently used nonlinear systems, i.e., *affine systems*[1], are considered and two neural network models, multilayer feedforward and radial basis function networks, are used to identify and control the nonlinear systems. As the most important results, the adaptive laws for adjusting the parameters of both of the two neural network models are derived using different methods and the stabilities of the overall systems are analyzed.

The objective of this chapter is to propose a design method, based on the work of [68], for identification and adaptive control of discrete-time single input single output nonlinear dynamical systems using multilayer feedforward neural networks. The identification scheme of our method is basically the same as that of [68], except that we use total random signals as the inputs to the neural network. As to the control part, a new scheme is proposed which greatly improves the abilities of the original method. In [68], the neural network for control is always a copy or inverse of the network for identification, which results in a control scheme that can only deal with the plants with the following known structure

$$y(k+1) = f[y(k), y(k-1), \cdots, y(k-n+1)]+$$
$$g[u(k), u(k-1), \cdots, u(k-m+1)], \tag{6.1}$$

in which y and u are the output and input to the plant, $f(\cdot)$ and $g(\cdot)$ are unknown nonlinear functions. Observe that the contributions from the input and the output are additive. In our scheme, the control inputs are not obtained directly from the identification network (or its inverse), but calculated through a nonlinear optimization process. In this way the plants with the following general form

$$y(k+1) = f[y(k), y(k-1), \cdots, y(k-n+1),$$
$$u(k), u(k-1), \cdots, u(k-m+1)] \tag{6.2}$$

can also be regulated.

[1] An affine system is defined to be a nonlinear system of the form

$$\dot{x} = f(x) + g(x)u,$$

where $u \in \mathcal{R}$ is the input, $x \in \mathcal{R}^n$ is the state and f, g are smooth vector functions.

The rest of the chapter is structured as follows. In the subsequent section the model of multilayer feedforward neural networks is introduced and the particular structure of the network as well as the two kinds of neurons used in this work are described. Also in this section a compact form of the back-propagation algorithm is derived which is easier to present and program, especially when using matrix based software packages, like Matlab. In Section 6.3, the model of using multilayer feedforward neural networks to identify nonlinear systems is introduced, based on which the optimization based control schemes are discussed. Section 6.4 presents the simulation results of the control system followed by the conclusions and additional remarks in Section 6.5.

6.2 Multilayer feedforward neural networks and back propagation algorithm

The function of a neuron in the network is two-fold, i.e., make a summation of the weighted outputs of the neurons connected to this neuron and pass the result through some simple functions, either linear or nonlinear. One of the typical nonlinear functions is the sigmoidal function

$$y = f(x) = \frac{1}{1+\mathrm{e}^{-x}}, \tag{6.3}$$

which is also one of the two functions used in our neural network model of identification and control for nonlinear systems. Neurons with the above nonlinear function will be called sigmoidal neurons. The other function used for the neurons in the net is a linear function, while neurons with such functions will be called linear neurons.

Now let us discuss the structure of the network for the identification and control of nonlinear systems. Based on others' experiences [68] and our own experiments, the neural net to be used will be a three-layer feedforward network with two hidden layers and a single neuron in the output layer (see Figure 6.1). The number of neurons in the two hidden layers are chosen somehow on an empirical bases and also according to the complexity of the plants, while the number of the input nodes are determined by the number of related past outputs and inputs of the system. This particular parameter of the

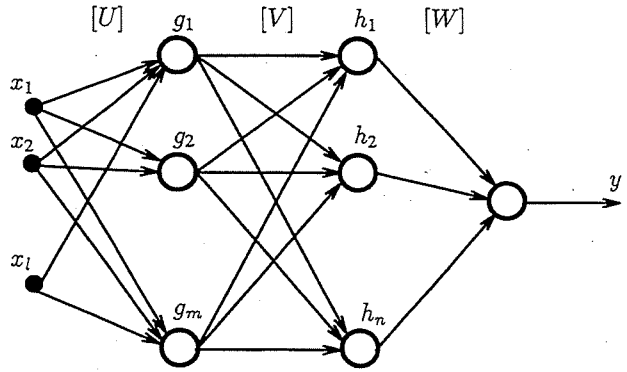


Figure 6.1: The structure of the neural net used in this work.

system should be known before hand, although it is not always the case. The details concerning how to chose the numbers of inputs and hidden neurons will be further discussed in the simulation section.

The two kinds of neurons (sigmoidal and linear neurons) discussed above are used in different layers of the network. Precisely, in the neural net shown in Figure 6.1, only the neuron in the output layer of the network is a linear neuron, all the rest are sigmoidal neurons. The reason for this arrangement is to stretch the functional range of the net to obtain any real value for the output of the neural net.

The neural nets for the identification and adaptive control of the nonlinear systems will be trained using back-propagation algorithms. Note the essential idea and details of the algorithm can be found in [56][84]. In the following we will derive a matrix form of the back-propagation algorithm suitable for the training of network shown in Figure 6.1. This compact form will make the program of the algorithm much easier especially when using some matrix based software package, e.g., Matlab.

Denote the inputs of the net by an l-column vector $x = [x_1 x_2 \cdots x_l]^T$, the outputs of the first hidden layer by an m-column vector $g = [g_1 g_2 \cdots g_m]^T$

and the outputs of the second hidden layer by an n-column vector $h = [h_1 h_2 \cdots h_n]^T$. Denote the real output of the network by scalar y and the desired output by y_d, recall that there is only one output neuron in this network. Upon assuming the number of inputs is l, the numbers of neurons in the first and second hidden layer are m and n, we further denote the weight matrix connecting the inputs to the neurons in the first hidden layer by an $l \times m$ matrix U, with its element u_{ij} $(i = 1,2,\cdots,l,\ j = 1,2,\cdots,m)$ being the connection weight between the ith input and jth neuron in the first hidden layer. The $m \times n$ matrix V and n-column vector W are defined similarly for the connection weights between the first hidden layer and the second hidden layer, respectively, between the second hidden layer and the output neuron.

As presented earlier, the nonlinear function of all the hidden neurons is a sigmoidal function as shown in Equation (6.3). Then it is not difficult to find that

$$f_x'(x) = f(x)[1 - f(x)].$$

Since

$$\begin{aligned} y &= W^T h \\ h &= f(V^T g) \\ g &= f(U^T x), \end{aligned}$$

using the chain rule of partial derivatives we can obtain the following:

$$\frac{\partial y}{\partial W} = h,$$

$$\frac{\partial y}{\partial V} = g \cdot W^T \cdot \begin{bmatrix} f'([V^T g]_1) & & \\ & \ddots & \\ & & f'([V^T g]_n) \end{bmatrix},$$

$$\frac{\partial y}{\partial U} = x \cdot W^T \cdot \begin{bmatrix} f'([V^T g]_1) & & \\ & \ddots & \\ & & f'([V^T g]_n) \end{bmatrix} \cdot V^T \cdot \begin{bmatrix} f'([U^T x]_1) & & \\ & \ddots & \\ & & f'([U^T x]_m) \end{bmatrix},$$

in which the partial derivative is applied with respect to each element of the matrices U, V and W. If we define the energy function to be

$$E = \frac{1}{2}(y_d - y)^2,$$

where y_d is the desired output of the network, we then have the weight update formulas in the matrix form

$$U(k+1) = U(k) + \eta \frac{\partial y}{\partial U}(y_d - y), \tag{6.4}$$

$$V(k+1) = V(k) + \eta \frac{\partial y}{\partial V}(y_d - y), \tag{6.5}$$

$$W(k+1) = W(k) + \eta \frac{\partial y}{\partial W}(y_d - y), \tag{6.6}$$

in which η is the gain term.

These formulas will be used to train the neural networks for the identification of the nonlinear systems as well as for the adaptive law for the neural adaptive controller.

6.3 Identification and control

The nonlinear dynamical systems considered in this work will be the so-called *bounded-in bounded-out* (BIBO) stable systems and are described by the following difference equation

$$\begin{aligned} y(k+1) = f[&y(k), y(k-1), y(k-2), \cdots, y(k-n+1), \\ &u(k), u(k-1), \cdots, u(k-m+1)], \end{aligned} \tag{6.7}$$

where $f(\cdot)$ $(\mathcal{R}^{n+m} \Rightarrow \mathcal{R})$ is a nonlinear continuous vector function, n and m are the number of delayed outputs and inputs, $u(k-i)$ $(i = 0,1,\cdots,m-1)$ and $y(k-j)$ $(j = 0,1,\cdots,n-1)$ are inputs, and respectively, outputs of the plant.

Although the knowledge about the structure of the plants, i.e, the specific form of function $f(\cdot)$, can definitely facilitate the design and the training of the neural identifier and controller, we will in this work assume that no prior knowledge about the model of the plant is available, whereas for the purpose of design and simulation, an input-output model (or so-called *black box*) of the plant is assumed available.

Another point that needs to be specified is that the number of delayed inputs m and outputs n which are relevant to the current output of system (6.7)

must be known *a priori* in order to determine the number of input nodes of the neural network which is $m + n$.

For the purpose of controlling (regulating, tracking, model reference control, etc.) the system described by (6.7), one method is to use a general mathematical model to represent the original plant. To qualify for such a representation, a mathematical model must be able to approximate any plants with the form (6.7), while the nonlinear mapping $f(\cdot)$ can be any continuous function. Apparently, this mathematical model should be a parameterized one and a set of parameters which together with the mathematical model satisfying certain approximation measurement must exist and be obtainable for any particular form of $f(\cdot)$.

A Multilayer feedforward neural network is a good candidate for such a mathematical model. Indeed, it has been proven recently [22][44] that a multilayer feedforward neural network with two layers of neurons (one hidden layer) can approximate any nonlinear mapping $f(\cdot) : \mathcal{R}^{m+n} \Rightarrow \mathcal{R}$, if the number of neurons in the hidden layer is sufficiently large. Precisely, if a multilayer feedforward network is represented by a mapping $N(l, w)$ where l is the number of neurons and w is the weight vector, then when l is sufficiently large, there will exist a w^* such that $N(l, w^*)$ approximates $f(\cdot)$ to any desired degree of accuracy. When an appropriate set of input-output data of $f(\cdot)$ is available, this w^* can be found using various learning algorithms, e.g., error back-propagation. For a detailed proof and exploration, the interested reader is referred to [22][44].

There exist other mathematical models suitable for the identification of nonlinear plants with form (6.7), but we will concentrate on a two-hidden-layer feedforward neural network model (see Figure 6.1) in this work. Standard back-propagation algorithm will be used to find the optimal weights, although other training algorithms may also be used and the result will stay the same.

6.3.1 Identifications

One method of using neural networks to identify nonlinear plants is the so-called *direct identification*, which means to train the network to identify the plant itself directly. This method is used in [68] and is further distinguished by parallel identification model and series-parallel identification model, following the results of the traditional system identification theory. Figure 6.2 shows the

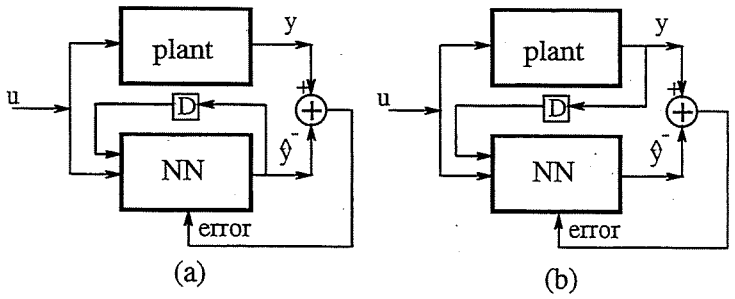


Figure 6.2: Direct identification. (a) Parallel model. (b) Series-parallel model.

structures of the two models.

In [31], we used a so called inverse identification model to set up a neural model of the nonlinear plant which makes the control of a certain class of nonlinear system much easier. The scheme of this inverse identification is shown in Figure 6.3. One disadvantage of this inverse identification scheme is that it can only deal with the class of nonlinear plants with the following form:

$$y(k+1) = f[y(k), y(k-1), y(k-2), \cdots, y(k-n+1), u(k)]$$

instead of (6.7). In this work the series-parallel model of direct identification will be used to identify the nonlinear plant.

6.3.2 Predictive control

Although it is normally a necessary pre-step for control, identification itself is not the final goal. It just provides an input-output representation of the unknown plant which will serve certain kind of control scheme. This representation should generate a reasonably accurate output of the plant whenever given an current input and the past relevant inputs and outputs.

Suppose this identified input-output relation is represented by

$$\hat{y}(k+1) = \hat{f}[y(k), y(k-1), y(k-2), \cdots, y(k-n+1),$$

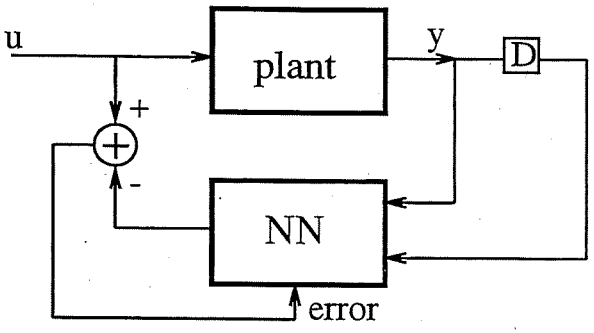


Figure 6.3: Inverse identification.

$$u(k), u(k-1), \cdots, u(k-m+1)] \tag{6.8}$$

where $\hat{f}(\cdot)$ is an approximation of $f(\cdot)$ in (6.7) by MLP. With the structure and the weights of the identification neural network obtained from learning process, $\hat{f}(\cdot)$ can also be looked upon as a known nonlinear function with all the weights as its parameters.

To simplify the presentation of the predictive control scheme, we assume that the control task here is to move the set-point of the plant from the initial value y_0 to the desired value y_d, while the result still holds true for other control tasks, e.g., tracking.

Suppose at time instance k the current output and the related past outputs and inputs are available, which are $y(k), y(k-1), \cdots, y(k-n+1), u(k-1), u(k-2), \cdots, u(k-m+1)$. One way to realize the control specification (i.e., bring the output of the plant from $y(k)$ to y_d) is to find such a current input $u(k)$ that minimizes the difference between $y(k+1)$ and y_d. More precisely, this control task is equivalent to the following optimization problem:

$$\min_{u(k)\in\sigma} |y_d - y(k+1)|, \tag{6.9}$$

or

$$\min_{u(k)\in\sigma} |y_d - f[y(k), y(k-1), y(k-2), \cdots, y(k-n+1),$$

$$u(k), u(k-1), \cdots, u(k-m+1)]|, \tag{6.10}$$

in which σ is the allowed range of $u(k)$. Since $f(\cdot)$ is the unknown nonlinear function of the plant and $\hat{f}(\cdot)$ is a neural network approximation of $f(\cdot)$, in practice we solve the following optimization problem instead

$$\min_{u(k)\in\sigma} |y_d - \hat{f}[y(k), y(k-1), y(k-2), \cdots, y(k-n+1),$$

$$u(k), u(k-1), \cdots, u(k-m+1)]|. \tag{6.11}$$

The range σ for $u(k)$ within which the object function is optimized plays an important role in this control scheme. Intuitively, $u(k)$ should be within a certain range of $u(k-1)$, so σ is chosen to be in the following range

$$\sigma = [u(k-1) - \Delta u, u(k-1) + \Delta u].$$

Although the specific value of Δu is chosen more or less on *trial-and-error* basis, there are still some guidelines available for that. First of all, in practice the difference between two consecutive control inputs can not be too large. Secondly, if Δu is too large which means the object function (6.11) has to be optimized within a big range w.r.t $u(k)$, the computation required for the optimization step will be too large, which makes the on-line working of the control scheme impossible. Thirdly, if Δu is too small which restricts the choices for $u(k)$, the value of the cost function will be reduced too slowly and hence results in a low speed of convergence.

Since (6.11) is a well defined nonlinear optimization problem, any standard nonlinear optimization algorithm can be used to solve it. Here we use a MATLAB function `fmin` to do the job. The algorithm `fmin` uses quasi-Newton method and `fmin(F,x1,x2)` returns the value of x that minimizes $F(x)$ in the interval $x1 < x < x2$. One of the common problems facing the existing algorithms (for non-convex optimization) is that these algorithms may end up with local minima. Fortunately this problem is not as serious as one would imagine for this particular nonlinear optimization problem of predictive control, which is due to the following facts: First, our control scheme is a dynamical process, and the output of the plant is brought to the desired value gradually by a series of control inputs which are generated by the iteration of the optimization process, while each step of the optimization is not necessarily

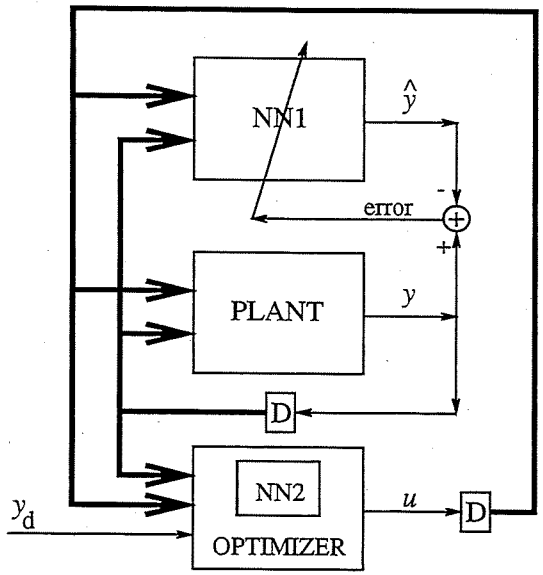


Figure 6.4: Optimization based predictive control scheme. D stands for multiple delay units and the wide line stands for multi signal flows. The initial set up of NN1 comes from the off-line identification and is further tuned by the 'error' signal, while NN2 is always a copy of NN1.

required to generate the best control input; Second, even for a certain step the algorithm ends up with local minima, it is almost impossible to stay there because the parameters of the objective function (related outputs and inputs) are changing from step to step.

As one may expect, the performance of this optimization based predictive control scheme is heavily dependent on the accuracy of the identification of the plant. To guarantee the convergence of the control scheme, an on-line identification of the plant is introduced in the control scheme to compensate the error and incompleteness of the off-line identification. The whole control scheme is illustrated in Figure 6.4. Due to the introduction of the on-line identification, the control scheme is also adaptive in the sense that it can deal with the parameter changes of the plant after the off-line identification.

6.4 Simulation results

In this section, we will present the results of extensive simulations on the identification and control schemes proposed in the previous section. Several plant models are considered, with each of them representing a broad class of nonlinear dynamical systems. All the simulations follow the same procedure, i.e.,

1. Proceed with the off-line identification of the plant using the model shown in Figure 6.2, in which the inputs $u(k)$ to the plant are random values with uniform distribution over $[-1, +1]$, while the difference between the output of the neural net and the real plant is used to update the weights of the net for 100000 times using back-propagation;

2. Plug the trained neural network into the optimization based predictive control scheme (Figure 6.4);

3. Provide a desired output of the plant y_d as one of the inputs to the optimizer;

4. Let the system evolve and the neural nets be updated until it converges.

Before starting the simulations, there are some parameters pertaining to the neural nets and back-propagation algorithm that need to be determined. One

of them is the number of hidden neurons in the network. It's easy to see that the number of network inputs should be the sum of the related past outputs and inputs of the plant, i.e., $n+m$. According to the experiences of others [68] and our experiment, the number of neurons in both of the first and second hidden layers are set to be $10 \sim 20$ for a plant with normal complexity. The other parameter that needs to be set is the back-propagation gain term η in (6.4), (6.5) and (6.6). Generally speaking, a bigger η will give a faster convergent speed but end up with a larger identification or control error, while a smaller η will just generate the opposite effect. In this work we tend to chose a smaller η because the off-line identification is allowed to last for a rather long period. A typical value of η is, say, 0.2.

Example 1.

In this first example, we will deal with a relative simple nonlinear plant, i.e., the input appears linearly with respect to the output, which is the Example 2 in [68]. The plant is described by the difference equation

$$y(k+1) = \frac{y(k)y(k-1)[y(k)+2.5]}{1+y^2(k)+y^2(k-1)} + u(k). \qquad (6.12)$$

Note that this equation is assumed unknown when carrying on the identification and control of the plant, while it is only used to generate the output of the plant when given an input and the delayed outputs. This will hold true for all the examples in the following. The control task for this and all the other examples is to make the plant tracking a desired output signal

$$y_d(k) = \sin(\pi k/100).$$

To proceed, we first do the off-line identification of the plant for 100000 iterations and the results of the last 500 iterations are shown in Figure 6.5(a). Then we plug the obtained neural network into our control scheme and try to make the output of the plant follow the prescribed signal. The result is shown in Figure 6.5(b).

To test the robustness of the control scheme, a simulation is done during which after 150 iterations the plant (6.12) was modified to

$$y(k+1) = \frac{y(k)y(k-1)[y(k)+1]}{1+y^2(k)+y^2(k-1)} + 1.5u(k). \qquad (6.13)$$

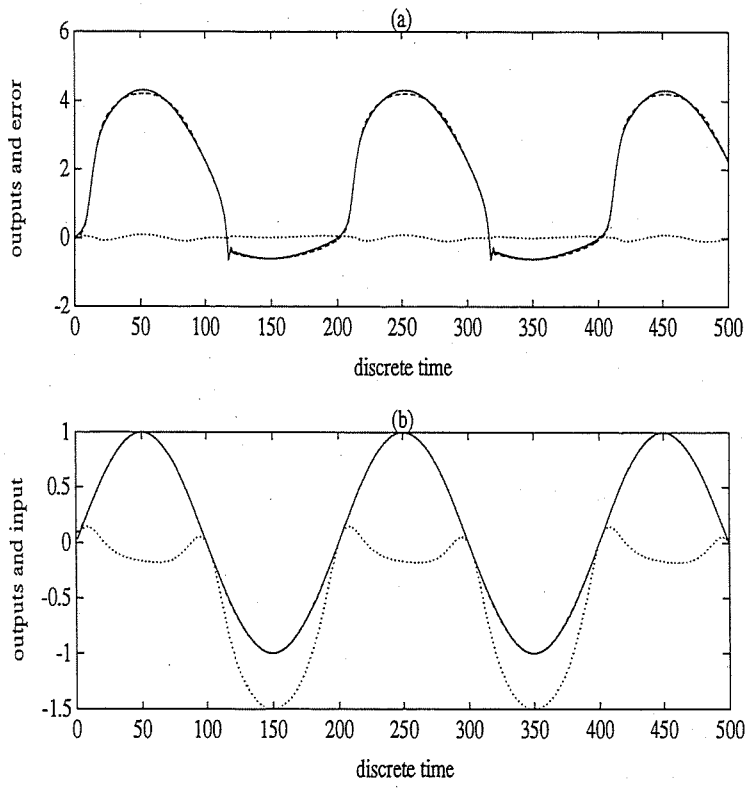


Figure 6.5: Simulation result of Example 1. (a) Result of off-line identification, solid line: desired output, dashed line: real output, dotted line: error. (b) Result of control, solid line: desired output, dashed line: real output, dotted line: control input.

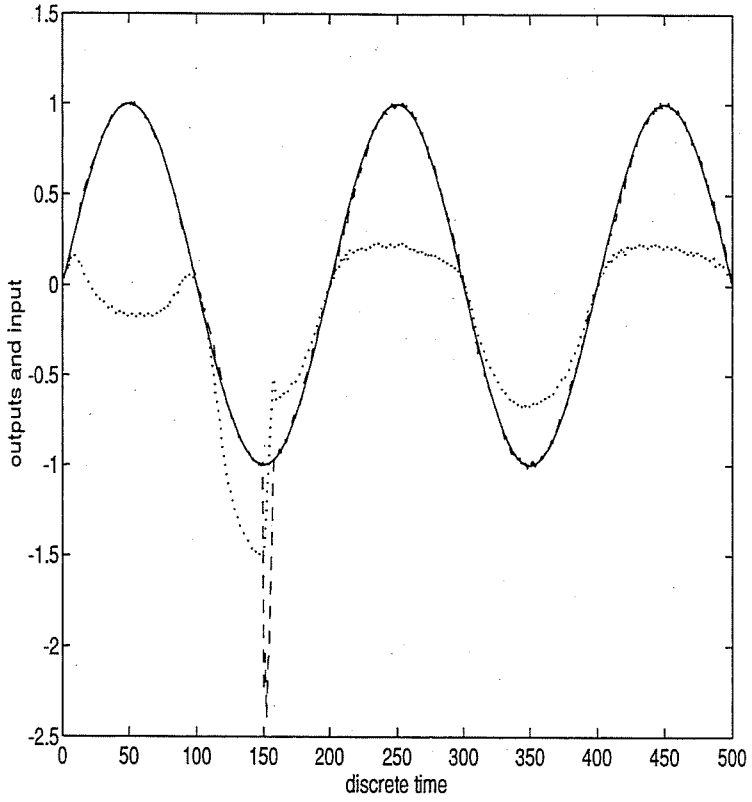


Figure 6.6: Simulation result of Example 1 when the plant was modified after 150 iterations, solid line: desired output, dashed line: real output, dotted line: control input.

The performances of the adaptive predictive controller is shown in Figure 6.6.

Example 2.

The second plant to be controlled is described by the difference equation

$$y(k+1) = \frac{y(k)}{1+y^2(k)} + u^3(k) \tag{6.14}$$

which is the Example 3 in [68]. Figure 6.7(a) shows the result of the adaptive control for 500 steps while (b) shows the behaviour of the controlled plant when it is changed to

$$y(k+1) = \frac{y(k)}{2+y^2(k)} + 2u^3(k) \tag{6.15}$$

after 150 steps of control.

Example 3.

For this example, we build up a nonlinear plant which is described by

$$y(k+1) = \frac{u(k)(\sin(y(k))+1.5)}{1+y^2(k)} \tag{6.16}$$

and the modified plant is described by

$$y(k+1) = \frac{u(k)(\cos(y(k))+0.5)}{2+y^2(k)} + 0.1. \tag{6.17}$$

The simulation result is shown in Figure 6.8.

Example 4.

In this last example, a much more complex plant with the general form of (6.7) is considered. The plant is the Example 4 in [68] and is described by the following difference equation

$$y(k+1) = \frac{y(k)y(k-1)y(k-2)u(k-1)[y(k-2)-1]+u(k)}{1+y^2(k-2)+y^2(k-1)}. \tag{6.18}$$

After 100000 iterations of off-line identification, the plant is still well controlled by the proposed adaptive control scheme. The result is shown in Figure 6.9.

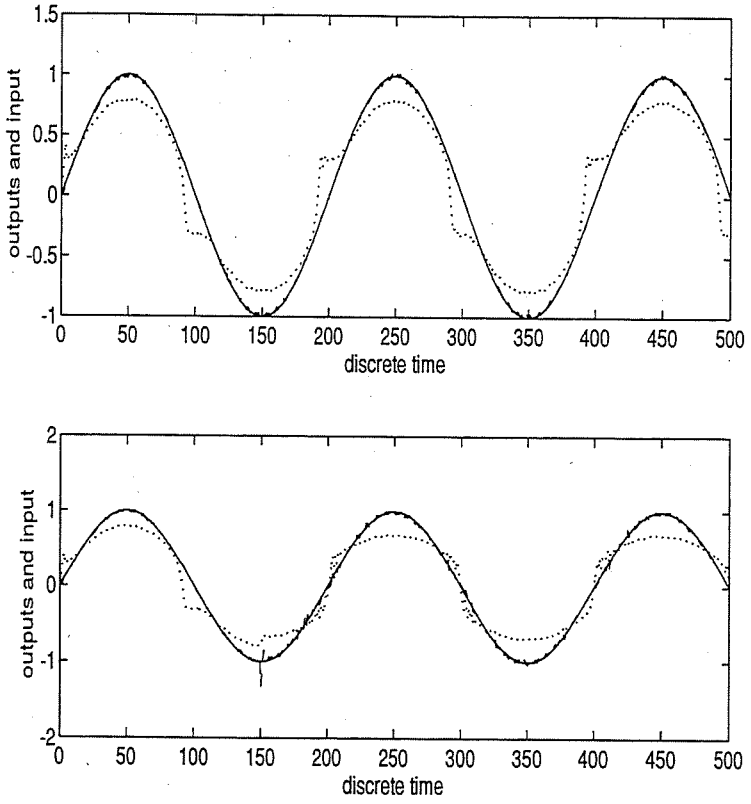


Figure 6.7: Simulation result of Example 2. The upper figure shows the result of control on the unmodified plant. The lower figure shows the result of control on a plant modified after 150 steps. Solid line: desired output, dashed line: real output, dotted line: control input.

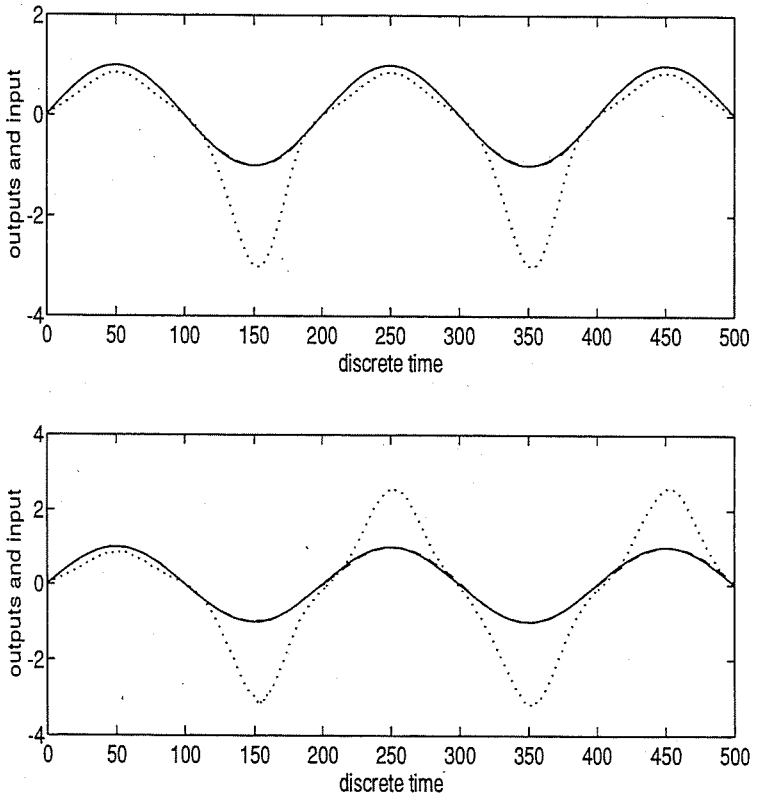


Figure 6.8: Simulation result of Example 3. The upper figure shows the result of control on the unmodified plant. The lower figure shows the result of control on a plant modified after 150 steps. Solid line: desired output, dashed line: real output, dotted line: control input.

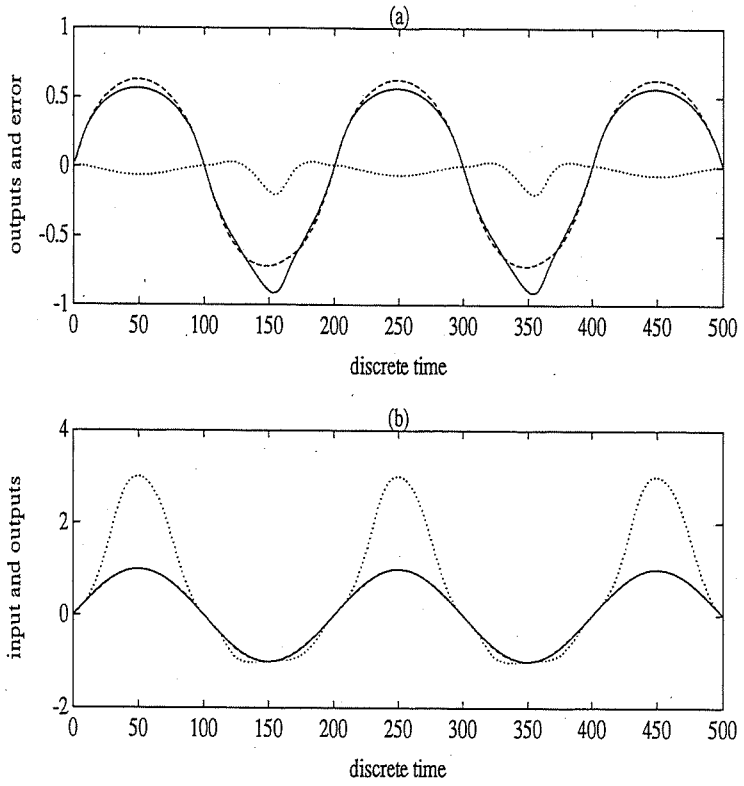


Figure 6.9: Simulation result of Example 4. (a) Result of off-line identification after 100000 iterations, solid line: desired output, dashed line: real output, dotted line: error. (b) Result of predictive control, solid line: desired output, dashed line: real output, dotted line: control input.

6.5 Discussions and conclusions

In this chapter we use multilayer feedforward neural network model for the off-line and on-line identification and control of a rather broad class of nonlinear dynamical systems. Based on the off-line identification, an optimization based one-step-ahead predictive control scheme is discussed. This adaptive control scheme, which combines the further on-line identification of the system with the control process, is proven to be effective and is especially preferable when the plant to be controlled is possibly changing its parameters during the identification and control process.

Although some promising results have been obtained for this neural network based identification and adaptive control of general nonlinear dynamical systems, this work is still in its preliminary stage. A great amount of work needs to be done to finalize and further develop the method. Future work may be directed to the following points:

- Work out a strict theoretical proof of the convergence of the proposed identification and adaptive control scheme, or find out and prove the conditions of the plants which the method can deal with;
- Generalize the method to the continuous nonlinear dynamical system;
- Develop the method for the multi-input multi-output systems;
- Try other neural network models, e.g., dynamical radial basis function network [79][73][74].

Chapter 7

Multi-Channel Signal Modeling by RBF Neural Networks

This chapter presents a neural network based recursive modeling scheme that constructs a nonlinear dynamical model for a given discrete multi-channel signal and the corresponding excitation. Using the so-called Radial-Basis-Function (RBF) neural nets as generic discrete-time nonlinear model structure and the ideas developed in the classical adaptive control theory, we have been able to derive a stable and efficient weight updating algorithm that guarantees the convergence for both the prediction error and the weight error. Elements of the spatial Fourier transform and sampling theory have been employed to provide the guidelines in choosing the parameters associated with the network structure and the RBF neurons. Simulation analysis is also carried out to highlight the practical considerations in using the scheme.

7.1 Introduction

In the area of digital signal processing, modeling is often used to establish a discrete-time dynamical model of a time series of interest. Such a model is then utilized for signal regeneration, prediction, analysis, feature extraction, and many other purposes.

To arrive at a closed-form model, a typical signal modeling scheme often involves a prior choice of a dynamical model structure. One of the conventional

model structures in wide-spread use is the familiar ARMA model and its variations. Useful as it has been proven though, the inherent linearity of the ARMA model structure gradually appears to be a limitation for adequate modeling of time signals generated from increasingly complex sources. There is, therefore, a well-founded reason to explore alternative model structures that are nonlinear.

Neural nets have recently attracted considerable attention for their potential role in offering alternative formulations and solutions for various nonlinear problems [46]. One of the key features of neural nets, relevant to the modeling problem, is the so-called universal approximation property possessed by classes of feedforward neural nets [74][22][44]. This property asserts that certain feedforward neural nets with finite number of neurons can be used to approximate arbitrary continuous functions defined on compact sets. The precision of the approximation is governed by both the network configuration and the weights in the neural nets.

The property of universal approximation has motivated the use of certain types of neural nets as generic nonlinear model structures. Similar to the linear case, the important issues in the neural-net-based nonlinear modeling are to select appropriate types of neural nets to serve as the model structure, and to devise algorithms that can find both the configuration and weights of a neural model *effectively* as well as *efficiently*.

In [68] an extensive effort has been made in using the back-propagation type of neural nets for nonlinear system modeling. Other recent works along the line have also emerged (see, for example, [46] for a concise survey). It appears that a common feature of these works is to apply directly to the modeling problem the existing weight-training and structure-setting techniques that have been developed in the general context of neural net design. As these techniques are often heuristic and do not take into account the particular nature of the nonlinear modeling problem, many of the resulting modeling methods based on them tend to be heuristic as well, and lack proper means for assessing such critical features as structural adequacy and convergence.

The objective of the present chapter is to demonstrate, through the development of a neural-net-based modeling scheme for multi-channel discrete-time signals, that the problem of neural-net-based signal modeling has its own particularities that can be explored for devising effective and efficient training

algorithms useful especially for solving the problem. In particular, we will develop a new modeling scheme by providing a specific formulation of the problem, and proposing a stable weight updating algorithm as well as a structure determination method within the formulation. There are three key ideas that underline the development of the new scheme.

First of all, the nonlinear model structure is chosen to be that of the so-called RBF neural nets. This choice is made not just because of the universal approximation property associated with this class of feedforward neural nets. More importantly, in the light of spatial Fourier analysis, RBF nets can be regarded as offering a general interpolating structure for function reconstruction using discrete data samples. This allows some rigorous techniques to be introduced in determining RBF neural net configuration. The linear-in-the-weights characteristics of RBF nets also facilitates the analysis and design of weight updating rule for the nets.

Secondly, the Lyapunov function approach developed in the framework of stable adaptive control theory is employed to derive a simple and efficient recursive algorithm for determining the weights of the RBF nets. This algorithm is proven to guarantee convergence for both the prediction error and the weight error.

Thirdly, the thoughts along the line of spatial Fourier transform and sampling theory are followed to examine the issue of RBF neural model structure determination. This includes setting the number of RBF neurons and their parameters. [85] has developed a specific method along this line. The problems with this method are revealed, and a modified version of the method is introduced in the present chapter that appears to be effective for common signal modeling tasks.

Let us briefly review the structure of RBF neural nets before proceeding to develop the new modeling scheme. A RBF neural net [8] is a two-layer feedforward network with the first layer of RBF neurons and the second layer of linear neurons as shown in Figure 7.1. Adjustable weights among the neurons only exist from the neurons in the first layer to those in the second layer. The inputs are simply directly connected to the neurons in the first layer via unity weights.

Referring to Figure 7.1, the ith RBF neuron in the first layer of a RBF net

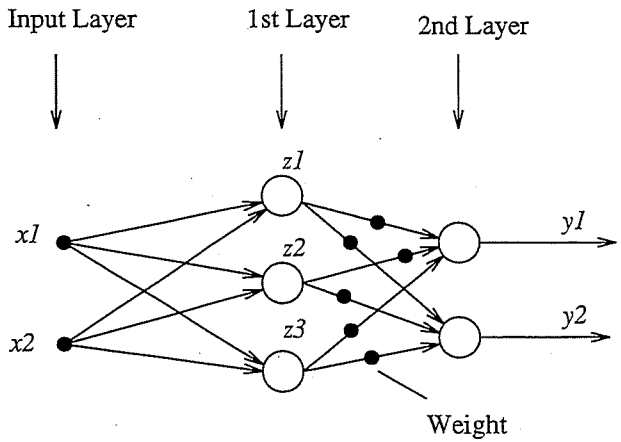


Figure 7.1: The structure of a RBF neural net.

has the following transfer function

$$z_i(x) = r(||x - p_i||/\sigma_i), \tag{7.1}$$

where $z_i(x)$ is the ith output of the neuron in the first layer. The argument x shows its explicit functional dependence on x which is the input vector of the form $x = [x_1 \quad x_2 \quad \cdots \quad x_q]^T$; $r(\cdot)$ is a radial basis function; p_i, σ_i (a positive scalar[1]) are, respectively, the center and the width for the ith RBF neuron. p_i, σ_i may be different for different neurons. Finally, the operator $||\cdot||$ denotes the standard vector norm. Using such common notations as R for the real field, R^+ for all the positive real numbers, R^n for the linear space of n-dimensional real vectors and $R^{n\times m}$ for all the real matrices of dimension $n \times m$, all the quantities appeared in (7.1) can be quantified precisely as $z_i(x) \in R, x \in R^q, p_i \in R^q, \sigma_i \in R$ and $r(\cdot) : R \to R$.

Denoting by y_i the output of the ith linear neuron in the second layer, and assuming that the number of RBF neurons is n, its transfer function is simply

[1] In general, σ_i need not be a scalar. It can be a vector or even a matrix as in [80] [53]

the weighted sum of the outputs from the first layer

$$y_i = \sum_{j=1}^{n} w_{ij} z_j(x),$$

where $y_i, w_{ij} \in R$. The above is clearly a linear operation. With m such linear neurons, it can be written in the following matrix form

$$y = Wz(x), \tag{7.2}$$

where $y = [y_1 \;\; y_2 \;\; \cdots \;\; y_m]^T \in R^m, z(x) = [z_1(x) \;\; z_2(x) \;\; \cdots \;\; z_n(x)]^T \in R^n$ and $W \in R^{m \times n}$ with w_{ij} being the ijth element of W.

For the sake of definiteness, we shall choose the RBF function $r(\cdot)$ to be the Gaussian function of the form

$$z_i(x) = e^{-||x-p_i||^2/\sigma_i^2},$$

where the parameters p_i, σ_i are termed, respectively, the center and the width of the Gaussian function. As n, the number of RBF neurons, p_i and σ_i together define the structure of a RBF net, they will be called structural parameters of the net. It should be noted that developed on the basis of Gaussian function though, the new modeling scheme to be detailed later (including the stable weight updating rule and the structure determination method) is equally applicable to other types of radial-basis-functions, like the inverse multiquadric RBF described in [60], as long as they are Gaussian-like and have the local activation nature.

Using the preceding notations, the universal approximation property of RBF neural nets can be stated specifically as follows. With an appropriate, but finite, number of RBF neurons in the first layer and proper setting of the parameters p_i and σ_i, there exists a weight matrix $\bar{W}$ such that the neural net with $\bar{W}$ can approximate, to an arbitrary precision, any continuous nonlinear function on a compact set (bounded and closed) [74]. This property is seen as a justification for RBF neural nets to be used as a generic model structure for nonlinear signal modeling. Another important feature of RBF nets is their linear-in-the-weights type of structure as depicted in (7.2). This structural feature allows us to separate the setting of the structural parameters n, p_i, and

σ_i from that of the weight matrix W. Such a separation considerably simplifies the development of the modeling problem.

The rest of the chapter is structured as follows. The subsequent section presents the general formulation of the signal modeling problem. It separates the modeling task into two subtasks of weight determination and structural determination, and develops a stable recursive weight updating rule for finding W. Section 7.3 considers the problem of structural determination. Some recent results on function approximation using spatial Fourier analysis are utilized to form a RBF grid with properly chosen widths in a hypercube where a given signal operates. Simulation analysis is carried out in Section 7.4 to illustrate the effectiveness of the scheme, and to provide many practical considerations in using it. Section 7.5 concludes the chapter by offering some additional discussions on the scheme.

7.2 The modeling scheme

7.2.1 General idea

Let a discrete-time multi-channel signal $\{y_t\} \in R^m \;\; (t = 0, 1, \ldots)$ be generated by an *unknown* discrete-time dynamical system of the following form

$$y_{t+1} = f(y_t, y_{t-1}, \ldots, y_{t-k+1}, u_t, u_{t-1}, \ldots, u_{t-l+1}), \tag{7.3}$$

where $u_t \in R^q$ is the excitation at time instance t; and $f : R^{km} \times R^{lq} \to R^m$ is a nonlinear vector function which is of course unknown; k, l are two positive integers relating the current sample to k samples and l excitation samples in the past. Note that u_t may be absent in many modeling situations. Its inclusion here only serves to make the problem formulation general.

The basic idea of our signal modeling scheme is to use RBF nets as a general model structure for the unknown nonlinear function $f(\cdot)$. It follows from the universal approximation property of RBF nets [74] that $f(\cdot)$, defined on a compact set $\mathcal{S}$, can be approximated to a given precision with an appropriate weight matrix and structural parameters. Or more precisely, assuming $x = x_t \stackrel{\triangle}{=} [y_t, y_{t-1}, \ldots, y_{t-k+1}, u_t, u_{t-1}, \ldots, u_{t-l+1}]^T$ and $y = y_{t+1}$ in (7.2), the universal approximation property asserts the existence of a weight matrix $\bar{W}$ and the structural parameters n (the number of RBF neurons), p_i and σ_i such

that a given signal $\{y_t\}$ along with its excitation sequence $\{u_t\}$ can be modeled as follows

$$y_{t+1} = \bar{W} z(x_t) \tag{7.4}$$

with x_t being confined to the compact set $\mathcal{S}$. (7.4) will be called a RBF neural model. Observe that in such a model, both the parameters n, p_i, σ_i involved in $z(x)$ and the weight matrix $\bar{W}$ are unknown. The central task of the signal modeling is, therefore, to determine n, p_i, σ_i and $\bar{W}$ so that (7.4) holds. Our approach is to separate the problem of determining $\bar{W}$ from that of the neural net structural parameters, and to develop separate methodology for their respective determination. Another requirement is that the modeling should be done recursively for the purpose of on-line application.

Translating the conditions required by the universal approximation into the specific ones for (7.3), we note that for the modeling problem to be well-defined [76], (7.3) needs to meet the following three conditions. First, as a discrete-time dynamical system, system (7.3) is required to be such that any excitation sequence and initial condition in a compact set[2] $\mathcal{S} \in R^{km+lq}$ will result in a signal sequence in $\mathcal{S}$. In other words, as long as (7.3) starts in $\mathcal{S}$, the subsequent dynamical iteration will not bring the output y_t out of $\mathcal{S}$. This condition can be interpreted as the well-known BIBO stability requirement in a restrictive sense. For a given bounded excitation sequence, it guarantees the existence of a compact region of operation $\mathcal{S}$ for (7.3). As will seen later, the existence of $\mathcal{S}$ is a key assumption for determining the RBF neural model structure. Secondly, $f(\cdot)$ should be continuous on $\mathcal{S}$. Finally, $f(\cdot)$ should be time-invariant, or what is the same, $f(\cdot)$ does not contain time-varying coefficients. This final condition ensures the well-definedness of the target weights in the constructed RBF neural model.

Note that k and l in (7.3) are the two parameters that are normally unknown for a particular modeling task, thus need to be determined first. Resembling the notion of degree in the ARMA models, these two parameters are very much linked to the physical nature of a signal, and can not be treated in the same way as we choose either W or n, p_i, and σ_i. In what follows, we will assume the knowledge of k and l in order to concentrate on purely neural net related

[2]Without loss of generality, in what follows we will assume the compact set $\mathcal{S}$ to be a closed hypercube or a hyper-rectangular in R^{mk+lq}. In both cases, $\mathcal{S}$ is symmetrical about the origin of the linear space R^{mk+lq}.

problems. Thus, neural net structure determination in this chapter is only meant to find the part of neural net structure defined by n, p_i, and σ_i given k and l. This confines the modeling problem to within a manageable scope. One should realize, however, that finding k and l is by no means trivial, and often requires area-specific knowledge, and/or well-developed heuristic methods.

Note also that due to the inevitable errors that may occur in determining both $\bar{W}$ and all the structural parameters n, p_i and σ_i, the signal model constructed by our scheme will not exactly resemble Equation (7.4), but will be in the following form

$$\tilde{y}_{t+1} = (\bar{W} + \Delta W)[z(x_t) + \Delta z(x_t)],$$

where $\Delta W, \Delta z(x_t)$ are the corresponding error terms. Because of this, the signal model built by the modeling scheme will have to be validated to ensure its applicability. There has apparently been no generally accepted methodology for validating a nonlinear signal model in a deterministic framework. Instead of attempting to develop such a framework, the present chapter will rely on the most straightforward method of checking the modeling error. The modeling error is defined to be the difference between $\tilde{y}_t$ generated by a RBF neural model and the real signal y_t of (7.3), both driven by the same excitation sequence and initial condition. We will be concerned with both the instantaneous modeling error at each time t, and the mean-square modeling error defined over a fixed number of time instances.

7.2.2 Weight updating rule

Let us consider the problem of determining the weight matrix first with the assumption that the structural parameters have been found. The methodology for structure determination will be described in the next section.

The idea for determining $\bar{W}$ is to construct a one-step-ahead recursive predictor which will generate $\hat{y}_t$, the prediction of y_t, using a linear dynamical system. An error model analysis will then yield a stable updating rule for the weight matrix.

To begin with, let $W_t \in R^{m \times n}$ be the weight matrix, and $e_t = \hat{y}_t - y_t \in R^m$ be the prediction error[3], both at time t. Upon introducing the following

[3]Note that the prediction error is different from the modeling error defined above in the sense

auxiliary error vector $\tilde{e}_t \in R^m$

$$\tilde{e}_{t-1} = \frac{Ce_{t-1} + W_{t-1}z(x_{t-1}) - y_t}{1 + z^T(x_{t-1})z(x_{t-1})}, \tag{7.5}$$

where $C \in R^{m\times m}$ is a constant diagonal matrix to be fixed later, a one-step-ahead predictor can be built as follows

$$\hat{y}_t = A\hat{y}_{t-1} - Ay_{t-1} + W_{t-1}z(x_{t-1}) - z^T(x_{t-1})z(x_{t-1})\tilde{e}_{t-1}, \tag{7.6}$$

where $A \in R^{m\times m}$ is a diagonal matrix with all its diagonal elements confined within $(-1, 1)$, implying that all the eigenvalues of A are strictly inside the unit circle.

Subtracting y_t from both sides of (7.6), and rearranging the terms give rise to the following error equation

$$e_t = Ae_{t-1} + W_{t-1}z(x_{t-1}) - y_t - z^T(x_{t-1})z(x_{t-1})\tilde{e}_{t-1}. \tag{7.7}$$

As will be shown shortly, the following weight updating formula

$$W_t = W_{t-1} - \tilde{e}_{t-1}z^T(x_{t-1}) \tag{7.8}$$

will make both the prediction error vector e_t and the weight error matrix

$$E_t \triangleq W_t - \bar{W}$$

converge to zero.

Although complex in its appearance, the idea behind the preceding updating rule is fairly straightforward. It simply realizes both the prediction and weight update as the convergence process of some stable dynamical system. If such a dynamical system is designed to have a unique stable equilibrium corresponding to the zero prediction and weight error, then its convergence dynamics will eventually settle down at the equilibrium yielding the desired prediction and the weight matrix. Naturally, the key step in implementing this idea is the proper construction of a stable embedding system. As will become

that the former is the error in the process of building up a RBF neural model. The latter is the error of the RBF model that has been built. Thus, the latter can be said to be the steady-state prediction error after convergence.

clear in the next subsection, both the auxiliary error $\tilde{e}_t$ and the prediction $\hat{y}_t$ are so constructed that they form a linear dynamical system with a nonlinear feedback. Combined with the weight updating formula (7.8), this system will be shown to have a single equilibrium that is asymptotically stable.

Before presenting a detailed analysis for the proposed updating rule, let us summarize the basic steps of our modeling scheme that can be followed to find a RBF neural model for a multi-channel signal $\{y_k\}$:

Step 1 Set k, l using the prior knowledge about the signal, or using some heuristic method. As stated in the preceding section, the problem of selecting these two parameters can not be resolved within the neural net framework itself, thus no systematic method is available for their determination.

Step 2 Choose the elements of the diagonal matrix A to be within $(-1, 1)$, and the diagonal matrix C in a way to be described in the next subsection; and set all the structural parameters n, p_i and σ_i of the RBF neural model using the method to be detailed later.

Step 3 Set the initial values $W_0, y_0, \hat{y}_0$ to prepare for the weight and prediction error updating. The choices that we have used for many of our testing examples are
W_0 = a matrix of random values between -1 and 1,
$y_0 = 0, \ \hat{y}_0 = 0.$
Note that this particular setting implies $e_0 = 0$. Other initial settings are equally valid. As will be seen later, provided that these values are finite, the proposed updating formulae will always converge to the unique equilibrium $E_t = 0, e_t = 0$ of the dynamical system defined by (7.5), (7.7) and (7.8).

Step 4 At time instance t, use (7.5) to compute $\tilde{e}_{t-1}$; then use (7.7) to compute e_t; and use (7.8) to compute W_t. Finally, compare e_t with a prescribed error bound ϵ.

Step 5 The iteration continues until $|e_t| \leq \epsilon$.

Step 6 When the iteration stops, the weights obtained can be used to build up the RBF neural model (7.4) for $\{y_k\}$.

In what follows, we shall provide a convergence analysis of the preceding weight updating rule, and discuss its properties in the light of the analysis.

7.2.3 Convergence analysis

We wish to prove that with appropriate choices for A and C, the preceding modeling scheme will indeed drive both e_t and E_t to zero. For notational convenience, the ith entry for the vectors $e_t, \bar{e}_t$ will be denoted by e_t^i, respectively, $\bar{e}_t^i \in R$, and the ith row of the matrix E_t by E_t^i, where $(E_t^i)^T \in R^n$.

To prove the convergence, let us observe that (7.7) and (7.5) can be regarded as constituting a discrete-time mth-order linear system in the following state-space form

$$\begin{aligned} e_{t+1} &= Ae_t + Bv_t \\ \bar{e}_t &= Ce_t + Dv_t, \end{aligned} \tag{7.9}$$

where the matrices A and C have been introduced in (7.6), respectively, (7.5); $B = D = I \in R^{m\times m}$, I is the unit matrix; and $v_t = E_t z(x_t) - z^T(x_t)z(x_t)\bar{e}_t \in R^m$ representing a closed-loop output feedback term. As all the constant matrices A, B, C, and D involved are diagonal, System (7.9) with the feedback term v_t can also be treated as a collection of m linear discrete-time first-order single-input and single-output systems of the following form

$$\begin{aligned} e_{t+1}^i &= a_{ii}e_t^i + v_t^i \\ \bar{e}_t^i &= c_{ii}e_t^i + v_t^i, \end{aligned} \quad i = 1, 2, \ldots, m, \tag{7.10}$$

where a_{ii}, b_{ii} are the ith diagonal entries for A, respectively, B, with the loop closed by $v_t^i = E_t^i z(x_t) - z^T(x_t)z(x_t)\bar{e}_t^i$, the the ith entry of the feedback vector v_t.

We first show that if (7.10) is strictly positive real, then it will be asymptotically stable, consequently, e_t^i and E_t^i will converge to 0 as $t \to \infty$. As (7.10) is assumed to be strictly positive real, the discrete positive real lemma can be applied, which results in the following set of equations (see, e.g., [40]).

$$\begin{aligned} a_{ii}^2\lambda_i - \lambda_i &= -\rho_i^2 - \gamma_i \\ a_{ii}\lambda_i &= \frac{c_{ii}}{2} + \nu_i\rho_i \\ 1 - \lambda_i &= \nu_i^2 \end{aligned} \tag{7.11}$$

where $\lambda_i, \rho_i, \nu_i \in R^+$, and $\rho_i \in R$.

Consider the following scalar function

$$V_t^i \triangleq 2\lambda_i(e_t^i)^2 + E_t^i(E_t^i)^T. \tag{7.12}$$

It is obviously positive and vanishes only at $e_t^i = 0$ and $E_t^i = 0$. The forward difference of V_t^i is easily calculated as

$$\begin{aligned} \Delta V_t^i &= V_{t+1}^i - V_t^i \\ &= 2(a_{ii}^2\lambda_i - \lambda_i)(e_t^i)^2 + 2a_{ii}\lambda_i e_t^i v_t^i + \lambda_i(v_t^i)^2 + E_{t+1}E_{t+1}^T \\ &\quad - E_t E_t^T. \end{aligned} \tag{7.13}$$

Inserting (7.11) into (7.13) and rearranging the terms, we have

$$\Delta V_t^i = -2[\rho_i e_t^i - \nu_i v_t^i]^2 - 2\gamma_i(e_t^i)^2 + 2\bar{e}_t^i v_t^i + 2E_t^i(\Delta E_t^i)^T + \Delta E_t^i(\Delta E_t^i)^T,$$

where $\Delta E_t^i = E_{t+1}^i - E_t^i$. Inserting (7.8) and v_t^i into the last three terms in the above equation

$$\Delta V_t^i = -2(\rho_i e_t^i - \nu_i v_t^i)^2 - 2\gamma_i(e_t^i)^2 - z^T(x_t)z(x_t)(\bar{e}_t^i)^2 \le 0. \tag{7.14}$$

The above derivation allows us to conclude that V_t^i is actually a Lyapunov function for (7.10) with the loop closed by v_t^i.

Returning to (7.9), consider the following scalar function

$$V_t = e_t^T P e_t + \mathrm{tr}\,(E_t E_t^T), \tag{7.15}$$

where P is a diagonal matrix with the ith diagonal entry being λ_i; $\mathrm{tr}\,(\cdot)$ denotes the trace of a matrix, which is the sum of the diagonal entries. V_t is obviously positive, and vanishes only at $e_t = 0$ and $E_t = 0$. By noting the following equation

$$V_t = \sum_{i=1}^m V_t^i$$

and using (7.14), the forward difference $\Delta V_t = V_{t+1} - V_t$ can be calculated as

$$\begin{aligned} \Delta V_t &= \textstyle\sum_{i=1}^m \Delta V_t^i \\ &= \textstyle\sum_{i=1}^m [-2(\rho_i e_t^i - \nu_i v_t^i)^2 - 2\gamma_i(e_t^i)^2 - z^T(x_t)z(x_t)(\bar{e}_t^i)^2] \\ &= -2\textstyle\sum_{i=1}^m [(\rho_i e_t^i - \nu_i v_t^i)^2] - 2\textstyle\sum_{i=1}^m [\gamma_i(e_t^i)^2] \\ &\quad - \textstyle\sum_{i=1}^m [z^T(x_t)z(x_t)(\bar{e}_t^i)^2] \le 0. \end{aligned}$$

We can therefore conclude that V_t is a Lyapunov function for (7.9), and consequently, the closed-loop system (7.9) is globally stable. The particular structure of RBF nets can actually allow us to conclude more than the global stability. In fact, as $z(x_t)$ can never be zero, ΔV_t will not be zero except at the single point $e_t = 0$ and $E_t = 0$. Consequently, $e_t \to 0, E_t \to 0$ $(t \to \infty)$, implying that $e_t = 0$ along with $E_t = 0$ is the unique asymptotically stable equilibrium of the closed-loop system (7.9).

To summarize, starting with finite initial conditions for e_t and E_t, the updating formulae described in the preceding section will guarantee both the prediction error e_t and the weight error E_t to converge to zero. Observe that this convergence property is intimately linked to the positiveness of the functions $z(x_t)$, without which we would not be able to conclude that E_t goes to zero as well.

The preceding proof indicates that both a_{ii} and c_{ii} will have to be chosen to make (7.10) positive real. Careful analysis shows that as long as a_{ii} stays within -1 and 1, it will not affect the positive realness condition. Thus the task reduces to choosing the parameter c_{ii} to ensure the positive realness of (7.10). For all the simulation examples to be given in the next section, we have fixed c_{ii} to be $-\frac{1}{2}$ for $i = 1, 2, \ldots, m$. This choice not only guarantees the required positive realness, but also seems to yield good transient response for a wide range of systems we have simulated.

Note that a_{ii} affects the convergence rate for both e_t and E_t. The general trends are that the convergence gets faster when a_{ii} is closer to 0, and slower when it is nearer ± 1. However, too small an a_{ii} may seriously affect the transient portion of the error convergence process.

7.2.4 Additional remarks on the weight updating rule

As mentioned before, the mechanism that has been used in deriving the stable weight updating rule can be understood as embedding the weight updating process into a linear dissipating system (7.9) plus a nonlinear dynamical feedback term. Along with the weight updating formula, the two parts constitute a stable nonlinear dynamical system. Both the zero prediction error and the zero weight error are designed to be the unique equilibrium point of this nonlinear system. The proof given in the convergence analysis part clearly shows that no spurious equilibria exist for the system. Therefore, the nonlinear system

should eventually settle down at its unique equilibrium for any arbitrary initial nonzero prediction error and/or weight error. This is a general strategy that underlines many existing stable updating schemes.

Observe that system (7.10) with the feedback term v_t^i also arises as the error model for certain SISO adaptive control systems [55]. Our preceding construction of the weight updating rule can be seen as yet another instance where such an error model emerges in a multi-dimensional setting.

It may have been realized that the whole part of the analysis leading to the weight updating scheme is based on the assumption that $z(\cdot)$ is known (recall that $z(\cdot)$ contains all the structural parameters n, p_i and σ_i of the net). As will be seen later, the technique to be presented in the ensuing section may result in erroneous $z(\cdot)$ that will bias (7.4) into the following form

$$\bar{y}_{t+1} = \bar{W} z(x_t) + \epsilon,$$

where ϵ represents part of the modeling error. This kind of bias may, to lesser extent, be caused by the noise on the signal and excitation data samples due to the measurement. It is certainly of interest to know the effect of ϵ on the convergent behaviour of the system. Close analysis shows that ϵ adds an extra term in the Lyapunov function (7.15), which may change the desired single-equilibrium dynamics of the entire system when it is higher than a certain threshold. This is the issue of robustness that can be explored further along an analytical line. At present our experimental results seem to indicate the relatively good robustness feature of the updating rule, although there has been no detailed analysis available yet. Better robustness may be achievable if we can embed the model in an appropriate dissipating nonlinear dynamical system. The success of this idea relies on an appropriate choice of a stable nonlinear dynamical system.

7.3 Determination of RBF neural net structure

We now discuss the problem of determining the three structural parameters involved in $z(\cdot)$: The number of neurons n, the centers p_i's and the widths σ_i's. After a short account of the existing heuristic methods, a theoretical basis for such a determination is briefly presented. A specific technique then emerges from the discussion.

7.3.1 Heuristic methods and their drawbacks

A commonly used heuristic approach to RBF neural net approximation suggests that n, p_i and σ_i are closely related to the concept of data clusters generated by x_t samples in R^{km+lq}. In this framework, n is interpreted as the number of data clusters within a pre-defined compact set, and p_i, σ_i as the center, respectively, the radius of the ith data cluster. The central task is, therefore, to devise an effective algorithm that can detect the data clusters given a sequence of data.

The idea of data clustering comes naturally when RBF nets are used for data classification. In this context, given data samples often imply clear separation boundaries among a limited number of data clusters. With prior knowledge about the number of data clusters (usually known from a classification task itself), the centers and the radii of these data clusters can be readily obtained using the well-known k-means clustering algorithm [64].

Our extensive simulation analysis has revealed, however, that the idea of data clusters is not relevant to the structural parameters of a RBF net in the case of nonlinear signal modeling. Generally, a given sequence $\{x_t\}$ is distributed evenly in a compact operation region of the underlying system, thus there are no clearly distinguishable data clusters that emerge from the given data samples. Running the k-means clustering algorithm for the data often results in the supposed centers of clusters wandering around their initial positions aimlessly. Moreover, we can even build $f(\cdot)$ in (7.3) in such a way that the centers of the constituent Gaussian functions are outside the compact set in which $\{x_t\}$ lies.

Our simulation experiments using $f(\cdot)$ built with the Gaussian functions also show that the approximation error is quite sensitive near the exact RBF centers of the function. A small distance away from the exact centers, the error is far less sensitive to the location of the centers. Thus, there seems to be of little use to fine-tune the centers without knowing exactly where they are, which is always the case for our modeling problem.

Yet another heuristic method uses the idea of covering for the determination of the structural parameters [75]. In this case, all the centers are taken as the data points themselves (or as in the case of [54], data points are used as the initial centers that are subsequently modified by the gradient descent type of algorithms), and are sufficiently spread in such a way that if they are considered

the centers of hyper-spheres with radius σ, any data sample x_t should at least be covered by one such hyper-sphere. This idea leads to recursive methods that allocate additional neurons whenever there is a data sample outside the region covered by the hyper-spheres induced by the previous centers [95]. Intuitively appealing though, theoretical backing for this type of methods is yet to be worked out. As a result, each specific construction needs to be verified by simulation.

7.3.2 Spatial Fourier theory based structure determination

Note that the nature of the discrete signal modeling problem is to use discrete signal samples to construct the underlying system. This obviously raises the question as to whether a given set of discrete signal samples can adequately represent the system function. This is the issue that all the heuristic methods fail to address.

Intuitively, all the data will have to be dense enough in a compact region $\mathcal{S}$ to allow the reconstruction of the function in between the data points. Drawing a similarity with the well-known situation of time signal reconstruction from discrete-time samples, exactly how dense the data samples should be is expected to be made precise using the idea of frequency domain analysis and sampling theory. The only difference is that spatial Fourier analysis[4] should be used instead as the nonlinear functions involved in the signal modeling problem are spatial rather than temporal. Carrying on with the analogue, one can expect that the reconstruction should be possible for spatially band-limited nonlinear functions with spatially sufficiently close data samples.

These intuitive thoughts are made rigorous in an interesting work reported recently [85]. By confining the approximation to a compact set $\mathcal{S}$, and using a sufficiently fine grid of RBF neurons, [85] shows that the approximation can be arrived at to within a given error bound using only finite number of function samples that determine the weights [85]. This result forms an important theoretical basis for finding the structural parameters of RBF neural nets in the context of nonlinear signal modeling. Thus, instead of trying to rely on heuristics or blind search, the problem of determining RBF neural net structure

[4] By "spatial Fourier analysis" we mean multi-dimensional Fourier analysis. The term *spatial* is used to highlight the dimensional difference.

can be thought of as setting up a sufficiently fine grid of RBF neurons with their centers located at the crossings of the grid, the widths chosen using a simple equation derived from the theory. When a sequence of given signal samples is sufficiently dense in $\mathcal{S}$, such a choice of RBF neural model structure guarantees the existence of a set of weights so that the approximation meets a pre-specified error requirement.

It follows from [85] that both Δ and σ can be found[5], given certain frequency characteristics of the unknown function $f(\cdot)$ and a so-called over-sampling parameter. The corresponding modeling error bound can then be obtained by evaluating some Fourier transform related functions. Theoretically important though, there are two reasons that invalidate the formulae for Δ and σ suggested in [85] for the modeling problem. First, they rely on certain prior knowledge about the unknown system, which is unavailable in practical modeling situations. Secondly, as indicated by our simulation analysis, these formulae do not normally give rise to low mean-square modeling error. In particular, the suggested formula for σ appears to yield considerably higher value than it should be for constructing a more accurate model.

To be pragmatic yet keeping in line with the spatial sampling theory, we have introduced certain amendments to help determine Δ and σ. The general rule is to set Δ smaller when the modeling error is required to be smaller. Smaller Δ corresponds to higher sampling rate, thus more accurate reconstruction. But too small a Δ also implies higher computation load. In the absence of any prior knowledge about a signal, a proper Δ can not be fixed in one-shot. It can only emerge from several rounds of refinement of an initial coarse choice. Thus, the way we follow is to set up an arbitrary coarse grid initially, and refine it step by step until the required modeling precision is reached. Such a refinement can be justified using the sampling theory cited above.

Once Δ is fixed, an empirical relationship between Δ and σ is found to be

$$\sigma = \frac{2}{3}\Delta \sim \Delta.$$

At present, this empirical formula can not be substantiated theoretically. Our justification for it along a heuristic line is as follows. σ specifies the degree

[5] As in [85], all the σ_i's will be chosen to be the same, and the common width will be denoted by σ.

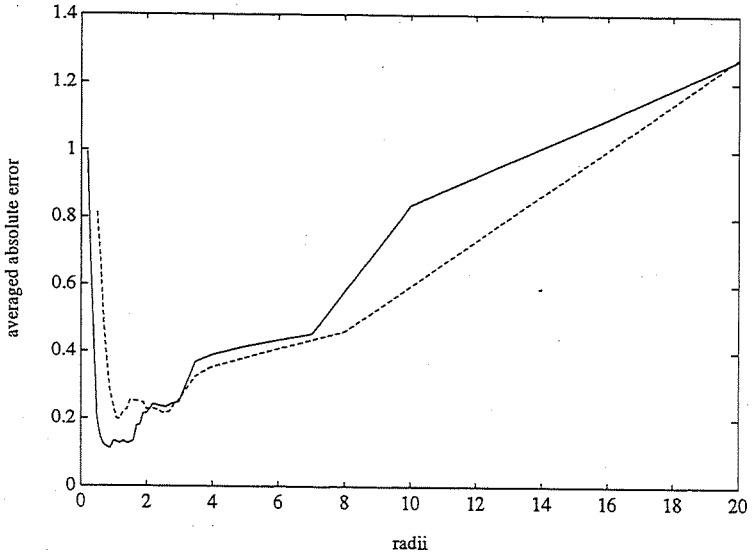


Figure 7.2: A simulation plot of mean-square modeling error versus the width σ. Solid line: A RBF net with 42 hidden neurons; dashed line: A RBF net with 13 hidden neurons.

of overlapping of effective radii for the RBF neurons with fixed centers. The overlapping obviously has a direct bearing on the model construction, as it signals how many neurons will cover a point in certain region of operation. Smaller σ limits the overlapping, thus localizing the model construction. But too small a σ may lead to gaps in between neurons. On the other hand, larger σ induces considerable overlapping, which translates into a more global model construction. Again too big a σ may result in the difficulty in balancing all the activated neurons, thus leading to large modeling error and excessive computation time. The proposed formula seems to have struck a balance between the two extreme cases.

These thoughts and, in particular, the empirical formula has been somehow verified by the extensive simulation we have performed. Our simulation has largely directed towards understanding the relationship between the choice of σ and the modeling error. Figure 7.2 shows the profile of the mean-square modeling error against the width that is normalized with respect to Δ for the modeling task set out in Example 2 (to be discussed in the next section). It is clear from the figure that for two different choices of the grid size, the formula tends to pick up a σ that makes the mean-square modeling error close to its minimum. The error profile shown in Figure 7.2 is typical of those appeared in other modeling problems we have simulated. This has served as a basis for us to draw up the formula. It is believed that a theoretical understanding of the formula should ultimately come from a careful spatial Fourier analysis.

Let us summarize the preceding discussions into algorithmic steps for the determination of all the structural parameters. Note that the modeling is confined to be within a compact hypercubic or hyper-rectangular set $\mathcal{S} \in R^{mk+lq}$. The existence of $\mathcal{S}$ is ensured by the BIBO stability of $f(\cdot)$, and by the bound on the excitation sequence $\{u_t\}$.

Step 1 Use $\{x_t\}$ to determine a compact set $\mathcal{S}$ to be the region of operation for an underlying generating system. $\mathcal{S}$ should obviously contain the origin of the linear space R^{km+lq}. The size of $\mathcal{S}$ can sometimes be fixed by the physical nature of a given signal.

Step 2 Form a uniformly spaced grid in $\mathcal{S}$, which is preferably symmetrical about the origin. The space between two neighboring grid points, denoted by Δ, can be adjusted to change the fineness of the grid, thus the modeling error of the RBF net. Once the grid is set up, the number of RBF neurons n is simply the number of grid points, and their centers p_i's are the coordinates of the grid points.

Step 3 Choose σ to be in the interval $\frac{2}{3}\Delta, \Delta$ as the widths for all the RBF neurons. The approximation error is closely related to the width in a way depicted in Figure 7.2. The optimal σ for a particular modeling problem normally takes a few rounds of trail-and-error to fix.

It is important to realize that although we assume the ideal noise-free signal samples for the sake of simplicity of the development, real measurement data are always corrupted by noise. Noise may completely change the frequency characteristic of a spatially band-limited nonlinear function $f(\cdot)$ so that a high sampling rate brings out not the new information about $f(\cdot)$ but the noise. Because of this, there appears to have an upper limit to the spatial sampling rate, thus the low bound for Δ. The exact nature of these bounds is yet to be worked out. Trail-and-error appears to be the only way to proceed for the time being.

Another practical issue concerns the density of the signal samples. Although we assume that the given signal samples are dense enough in a compact domain $\mathcal{S}$, there is no way that this can be ascertained without knowing the underlying system (unknown in all real modeling situations). This issue of data density is further complicated by the fact that unlike a static function approximation problem where the domain of a nonlinear function can be sampled anywhere and arbitrarily dense, the dynamical nature of the underlying system often prevents the adequate sampling in certain areas in the range of $f(\cdot)$. Lacking sufficient occurrence of signal samples in these areas, the proposed recursive weight updating algorithm can not adequately adjust the weights for accurate function representation in these areas. Consequently, the instant prediction error might momentarily shoot above the mean-square prediction error. This phenomenon of the so-called local spikes can occur even if the weight updating process shows convergence in averaged error, and all the structural parameters have been fine-tuned. A simple remedy for this is that once a proper Δ is set, the weight updating rule will be turned on to adjust the weights all the time, or whenever there appears to have significant modeling error for the testing signal. This mode of operation allows the updating rule to accumulate the

occurrence of the signal samples till the critical sampling density is reached. As we will see in the ensuing section that prolonged training will help to ease the problem of spikes to a considerable extent.

7.4 Simulation analysis

We now apply the modeling scheme developed in the preceding sections to three specific signal modeling problems. The simulation examples serve the dual purpose of showing the effectiveness of the scheme as well as providing many practical considerations in actually using it for modeling problems. To simplify the discussion and the illustration, the first two problems will only be concerned with modeling single-channel signals. Multi-channel signal modeling is presented as the last example. As will be seen, the multi-channel case generally exhibits similar convergence and error trends. However, the tedious nature of multidimensionality tends to make the simulation analysis less appealing.

Example 1

In this example, the proposed scheme is used to model a time series $\{y_t\}$ generated by driving the following nonlinear dynamical system

$$y_{t+1} = \frac{y_t}{1+y_t^2} + u_t^3, \tag{7.16}$$

with a random discrete-time sequence $\{u_t\}$ uniformly distributed in $[-1.1, 1.1]$. This system obviously satisfies the conditions for the scheme to be applicable. Note that (7.16) has also been considered in [68]. While the neural modeling technique used in [68] needs the following structural knowledge

$$y_{t+1} = N_f(y_t) + N_g(u_t)$$

about the system ($N_f(\cdot)$ and $N_g(\cdot)$ are some nonlinear functions), our scheme proceeds by using nothing more than $\{y_t\}$ and $\{u_t\}$.

The first step of the modeling scheme is to determine the operating region of the signal. By exciting the system with the random excitation sequence mentioned above, we can easily obtain the range of the output sequence to be

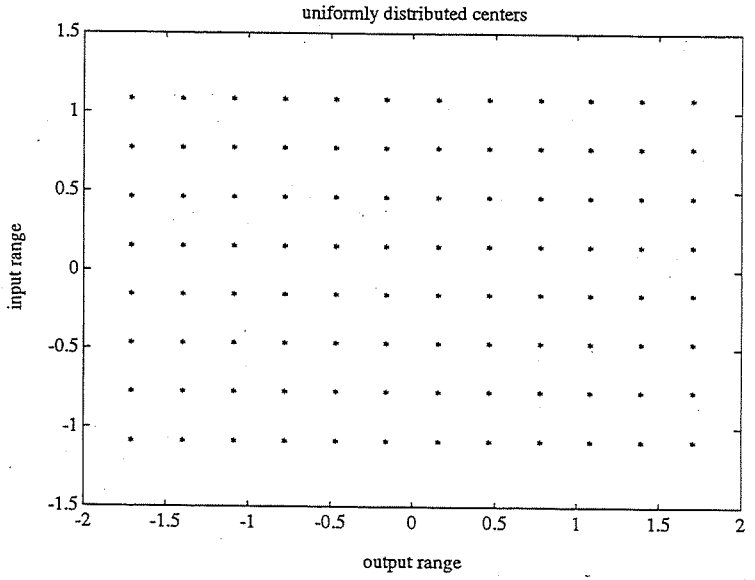


Figure 7.3: The grid showing the locations of the centers in *Example 1*. The operating region $\mathcal{S}$ is the rectangular $[-1.8, 1.8] \times [-1.1, 1.1]$. Here Δ (the horizontal or vertical distance between two neighboring centers) is chosen to be 0.31.

$[-1.8, 1.8]$. This, along with $[-1.1, 1.1]$, defines the compact set $\mathcal{S}$. Note that $\mathcal{S}$ in this case is a rectangular in the 2-D space (refer to Figure 7.3).

To keep the number of RBF neurons relatively low, and yet maintain the modeling error bound (set at $\epsilon = 5 \times 10^{-2}$ for this example), we choose Δ to be 0.31. This choice is made only after a few rounds of refinement. Once Δ is known, a grid can be evenly laid down inside the rectangular $\mathcal{S}$ with the grid points giving the number of neurons and the locations of the centers. There are 96 centers generated for this particular example, which are shown in Figure 7.3 to be uniformly distributed inside $\mathcal{S}$. Experience also indicates that modeling errors can also be reduced if the grid is chosen to be symmetrical with respect to the origin of the hypercube $\mathcal{S}$ as shown in the figure. The exact reason for this is not yet clear.

Using the empirical formula $\sigma = \frac{2}{3}\Delta$, the radius σ can be obtained to be 0.21. With all the centers and the radii fixed, the RBF neural net is trained with $\{u_t\}$, and $\{y_t\}$ which is generated on line by using $\{u_t\}$ as the excitation sequence to the system (7.16) and $y_0 = 0$. It is worth noting that the range of the excitation sequence should always be maintained inside $\mathcal{S}$ throughout the entire training process in order not to violate the stated conditions.

After a few thousands of weight updates, the mean-square prediction error e_t converges to within 0.02 (maximum instantaneous error at about 0.2) as shown in Figure 7.4. Observe the occasional error spikes that appear in the convergent process. These error spikes may lead to momentarily higher error even if the mean-square prediction error approaches zero. As discussed at the end of the preceding section, this phenomenon is due to the insufficient occurrence of signal samples in certain areas in the rectangular $\mathcal{S}$. Particularly, in those areas where $\{y_t\}$ has sharp turning points, or attains local extrema. Error spikes can be alleviated by prolonging the weight updating process to allow adequate appearance of all kinds of signal samples. This is clearly shown in Figure 7.4 where the mean-square prediction error at about 200 updates is not very much different from that at 5000. But the drop in local spikes is fairly visible indeed.

To test the constructed RBF neural model, we excite both the real system and the RBF neural model with the following periodical time series

$$u_t = \sin(2\pi t/25) + \sin(2\pi t/10)$$

and the same initial condition. Note that u_t is confined within the rectangular

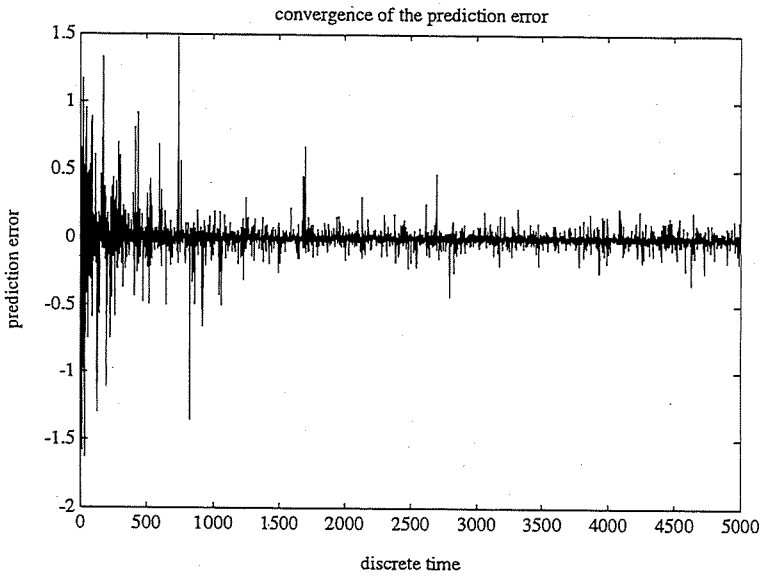


Figure 7.4: The convergence of the prediction error in *Example 1*.

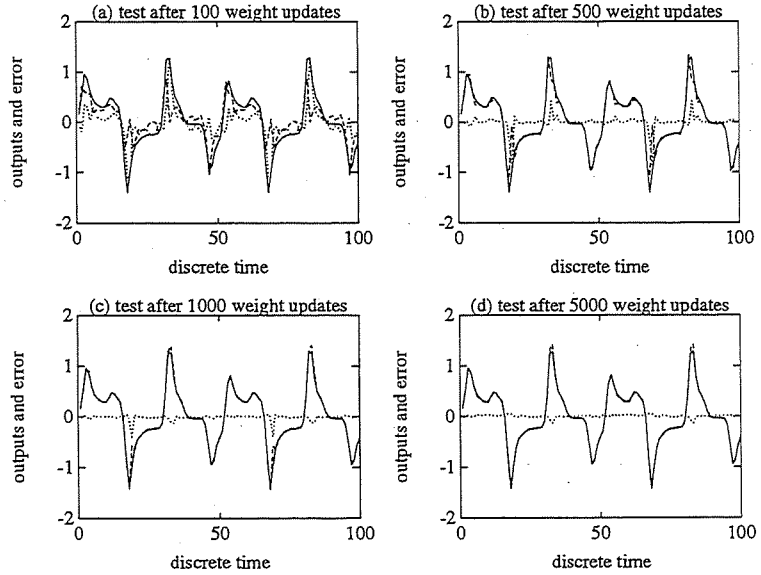


Figure 7.5: The result of modeling in *Example 1*. Solid line: Real signal; dashed line: Modeled signal; dotted line: Modeling error.

$\mathcal{S}$. Figure 7.5 shows the real signal, the signal generated by the model, and the modeling error. The tests have been performed for four different phases of the weight updates with the first test after only 100 weight updates, the second test after 500 weight updates, third test after 1000 weight updates, and the final test after 5000 weight updates. The testing results clearly indicate the improved modeling accuracy when the mean-square prediction error gets closer to zero. Note that 5000 weight updates for this particular example only require about tens of seconds in real time computation. This great efficiency can not be matched by the Back-propagation type of nonlinear identifiers.

Example 2

The signal in this second example is also generated using a nonlinear system given in [68]:

$$y_{t+1} = \frac{y_t y_{t-1} y_{t-2} u_{t-1}(y_{t-2} - 1) + u_t}{1 + y_{t-2}^2 + y_{t-1}^2}.$$

This underlying system has multiple delays, and its region of operation $\mathcal{S}$ can be determined as the hypercube $[-1, 1]^5$ in the 5-D space for random excitation sequences bounded within $[-1, 1]$.

Considering that the centers will have to be placed in the 5-D hypercube, we need to choose Δ relatively larger in order to keep the number of RBF neurons to be smaller. Yet ensure low modeling error (given as $\epsilon = 5 \times 10^{-3}$ for this example), Δ can not be too large. Our extensive simulation shows that setting Δ to be 0.9 appears to be a balanced choice. In this case, there will be the total of 243 centers needed to set up a uniform grid in the 5-D hypercube. Moving Δ downward to 0.6, the number of neurons required goes up sharply to 1024 with only little improvement in the modeling error. When Δ is set at 0.5, the number of neurons required ends up at 3125, so prohibitive a number that the weight updating is considerably slowed down. All these seem to indicate that the higher dimensional modeling problems, in general, may need more RBF neurons to achieve the required modeling accuracy. However, Δ in these cases can often be explored to strike a balance between the number of neurons and the modeling precision.

Following the empirical formula, the radius is set to 0.6. Training the RBF network with these structural parameters for 3000 iterations, the mean-square prediction error is shown to converge in Figure 7.6.

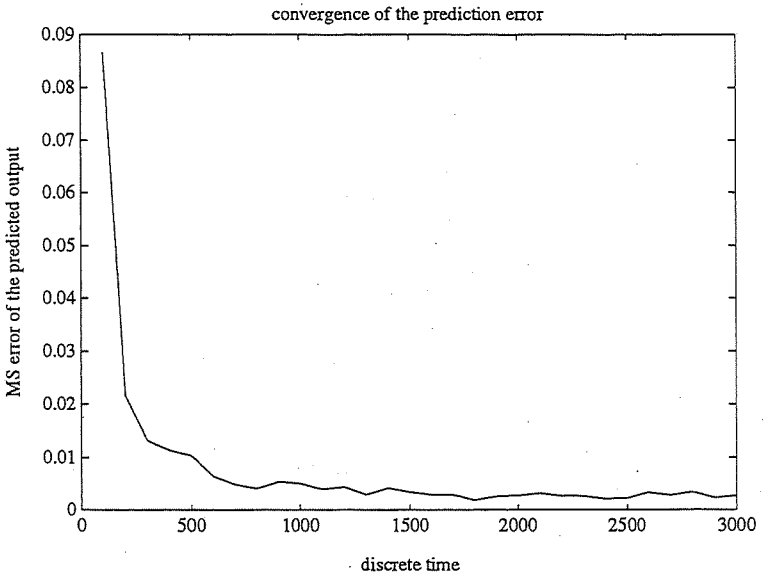


Figure 7.6: The convergence of the mean-square prediction error in *Example 2*.

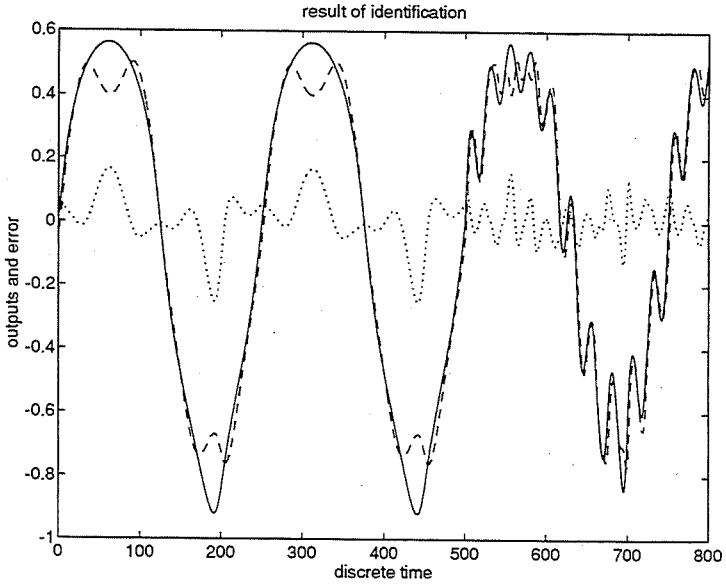


Figure 7.7: The result of modeling in *Example 2*. Solid line: Real signal; dashed line: Modeled signal; dotted line: Modeling error.

The model is constructed after about 3000 weight updates. To test the model, a periodic excitation signal

$$u_t = \begin{cases} \sin(2\pi t/250) & \text{when } t \leq 500 \\ 0.8\sin(2\pi t/250) + 0.2\sin(2\pi t/25) & \text{when } t > 500 \end{cases}$$

is applied to both the real system and the RBF neural net model. The two signals as well as the modeling error are shown in Figure 7.7. Observe the shear saving achieved by our scheme in terms of the number of iterations as compared to the Back-propagation based method (about 100,000 iterations are required as reported in [68]). Moreover, the number of weights involved in our scheme is slightly less compared to that in the Back-propagation net (243 versus 310).

Example 3

In this last example, a two-channel signal modeling is considered. The signal is produced by the following pair of nonlinear difference equations (again taken from [68])

$$\begin{bmatrix} y_1(t+1) \\ y_2(t+1) \end{bmatrix} = \begin{bmatrix} \frac{y_1(t)}{1+y_2^2(t)} \\ \frac{y_1(t)y_2(t)}{1+y_2^2(t)} \end{bmatrix} + \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix}. \tag{7.17}$$

Exciting the system with a sequence of 2-D random inputs uniformly distributed between -1 and $+1$, the output signal is readily determined to be in the range of $[-3.5,\ \ 3.5]$ for y_1, and $[-2.5,\ \ 2.5]$ for y_2. Choosing Δ to be 1 and σ to be 0.8, we end up with 432 neurons uniformly located in the 4-D operating region.

After around 20,000 weight updates, the predicted error converges to within the mean-square error of 0.06. Note that the number of updates is relatively large compared to the previous two examples (though this only amounts to about a few minutes in real time in a Unix Workstation), due mainly to the larger number of RBF neurons involved in the model. To test the modeling accuracy, the following periodic excitation

$$u(t) = \begin{bmatrix} \sin(2\pi t/25) \\ \cos((2\pi t/25) \end{bmatrix} \tag{7.18}$$

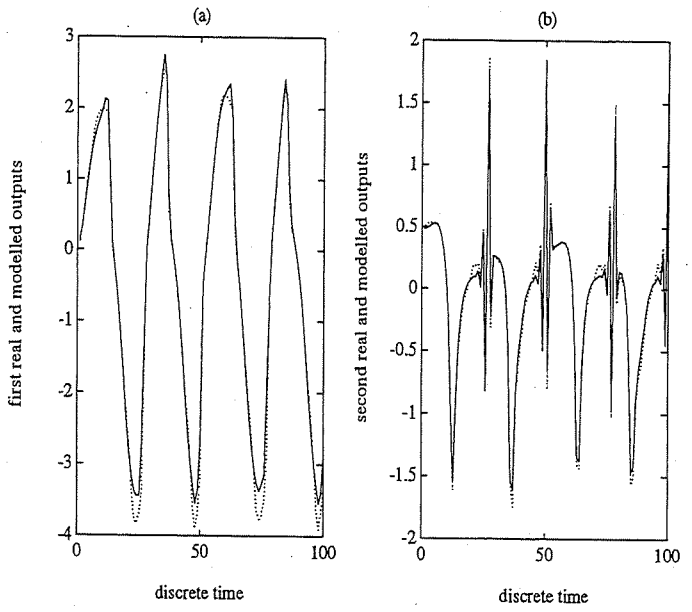


Figure 7.8: The result of modeling in *Example 3*. (a) is for the first channel and (b) is for the second channel signal. Solid line: Real signal; dotted line: Modeled signal.

is applied to both the real system and the RBF model with the real two-channel signals and the corresponding modeled signals shown in Figure 7.8. This example demonstrates the efficiency and accuracy of our modeling scheme as applied to higher dimensional modeling problems. As multi-channel and high dimensionality necessarily involve large number of neurons in the training, the computation time in this case is often considered to be a bottle-neck for practical applications. By offering a simple yet stable modeling scheme, we seem to have provided at least a partial solution to the problem.

7.5 Discussions and conclusions

We have presented a discrete-time nonlinear signal modeling scheme based on neural nets. The key motivation is to carry over the essence of the linear ARMA modeling to the nonlinear case with the help of RBF neural nets. In this particular framework, the modeling problem is decomposed into two separate sub-problems of determining the weights and the structural parameters. The theory of spatial sampling and the Fourier analysis helps to find the structural parameters in the light of function approximation. Once the structure is known, a stable recursive algorithm can be readily used to find the weights of the nets. The convergence of the algorithm is relatively easy to establish compared to other types of neural nets where convergence often comes as an accidental by-product of a prolonged training process.

Compared to many existing modeling techniques, such as the Back-propagation net based techniques, the present one has the clear advantage of accuracy-efficiency tradeoff, with the typical training cycle of a few thousands to reach to within the mean-square modeling error in the order of 1%. Moreover, compared to other types of RBF net based modeling techniques, each of the basic steps of the present modeling scheme is backed by theoretical considerations rather than pure heuristic considerations.

The use of RBF neural nets is often hindered by the perception that a RBF net would necessarily involve a great deal more neurons (thus more weights) than other kind of neural nets (*e.g.*, a Back-propagation net) for exactly the same approximation problem. Our experiences with the scheme indicate, however, that this needs not be the case, especially in terms of the number of weights for achieving the same accuracy. The examples given in the preceding

section have partially illustrated this point.

An important practical issue is to examine the effect of measurement noise of the signal and the excitation to the proposed scheme. This has only been briefly mentioned in conjunction with the structural bias without further analysis. Seeing from the experience with the scheme, it appears that the proposed scheme can absorb the noise to a certain extent. But a rigorous analysis needs to be done for a deeper understanding of the issue. For its nature, such an analysis may have to be built on a probabilistic setting rather than the present deterministic one. This is one of the current research direction we are pursuing.

As a final point, we note that the structural determination and the weight adaptation are treated as two separate problems. Especially, the former is done inherently off-line and in one-shot. For more effective nonlinear modeling, there is a recent trend to formulate the modeling problem in such a way that the model structure will also be generated gradually along with the associated weights. With all their structural features, RBF neural nets seem to be ideal for this kind of modeling operation.

Chapter 8

General Conclusions

In this dissertation we have discussed some basic theoretic problems and application perspectives of two types of neural networks, i.e., Hopfield neural networks and feedforward neural networks. The theoretic problems dealt with in this dissertation are the synthesis of feedforward multi-layer perceptron neural networks using geometric approach, improved and new learning algorithms for Hopfield associative memories, feedforward model of neural associative memories and cellular neural networks used as associative memories. For the application part, we talked about applications of neural networks to associative memory, rule-based nonlinear system control and the application of feedforward multi-layer perceptron networks and radial basis function networks to the general problems of nonlinear system identification (signal modeling) and control.

8.1 Direct design method for MLP

The design of a multi-layer perceptron network has been traditionally fulfilled within two steps. In a first step, the structure of the MLP for a particular problem (here in this dissertation we consider the dichotomy problem), i.e., the number of layers and the number of neurons in each layer, are determined based on the past experiences if there is any, or *ad hoc* method, or are just randomly chosen without any reason. With such a predetermined structure, some learning algorithm (e.g., error back-propagation) is then used on the learning samples to

train the weights of the network. Apparently, these procedures can not always lead to a good solution since on the one hand, the predetermined structure may be inappropriate or even incapable to solve that particular problem, and on the other hand, most of the currently available learning algorithms need to be fine tuned to perform best on a particular problem. As a result, the *trial-and-error* type of methods are needed here to come up with an admissible result. Further, since the ranges of the adjustable parameters of a network structure (number of layers and number of neurons in each layer) are rather large and for each setting of these parameters the learning algorithm should be applied to converge to a suboptimal set of weights, this trial-and-error procedure may take quite a long time to end up with a suitable network structure and weights for that particular problem.

There are two major approaches to design a multi-layer perceptron for practical tasks. The aforementioned trial-and-error method for the structure and learning for the weights is called the indirect method. The other one, which is called the direct method, uses direct setting strategies which are inspired by the analytic and geometric considerations to determine the network structure and the weights altogether in one shot. One of the drawbacks suffered by the indirect approach, as we have discussed, is that the determination of the structure is based on trail-and-error, and there are no general guidelines available. This often results in the over or under determination of the structures. Another drawback is that it generally takes a long time for the learning algorithm to converge, or even more seriously some learning algorithm is intrinsically not able to find the weights for certain dichotomy problems (see, for example [7]).

The direct approach can avoid the aforementioned problems, but will generally require the detailed understanding of the nature of both the problem and the net structure. The geometrical approach we have proposed in the first part (Chapter 2) of this dissertation belongs to this second category. Obviously, the detailed geometrical analysis has enabled us to come up with a quite systematic method for the determination of the net structure and the setting-up of the weights. It should be noted that our net structure is aimed at arbitrary problems, and a much smaller net might be obtained by exploring the nature of some specific problems. Obviously, the reduction is achieved at the expense of losing the generality. In this sense, our net structure provides a meaningful upper-bound (or more precisely, the maximum necessary) to the multilayer

perceptron used for this purpose.

Although we have taken the binary values as the inputs of the dichotomy problems in our discussion, the binary assumption on the inputs is not as restrictive as might be originally thought. Since for non-binary inputs, a preprocessing layer of neurons can always be added to map them into the vertices of a certain dimensional hypercube [63]. Besides, this preprocessing layer also provides the basis to the so-called *generalizing ability*.

8.2 Neural associative memories

In the second part (Chapter 3 and Chapter 4) of this dissertation, several problems are discussed around the topic of artificial neural network models of associative memories. Starting from the traditional network structure for associative memories (Hopfield network) and the widely used learning algorithm (Hebbian rule) we developed two new learning algorithms for the Hopfield net (fix point programming using back-propagation algorithm and optimization based learning rule), modified the structure of the Hopfield net (Hopfield net with hidden structure) in order to improve the performance, discussed the possibilities of using cellular neural network as a model of associative memories and finally we proposed a feedforward neural network model for associative memories and binary pattern associations.

Chapter 3 of this part deals with the problems of improving the performances of Hopfield associative memories by introducing new weight setup rules and modifying the standard Hopfield network structure, and also the problem of using cellular neural network as a model of associative memories.

Associative memory is one of the most promising potential application areas of artificial neural network owing to the parallel computation nature and the convergent dynamical behaviour associated with some models of neural networks. Initiated by J.J. Hopfield in his two famous papers [42] and [43], a model of fully connected neural networks (called Hopfield net) has been extensively studied to function as associative memories. Most of the previous work has focused on the properties and performance of the nets designed using the well known Hebbian rule ([59][17][102][65]) as well as the development of some other design rules (see [61][19] for some examples).

Under the so called *performance oriented* formulation of the design problem of neural associative memory, a data-driven approach has been proposed to design a memory with deterministic requirements on the fixed points or the size of attraction basin. First, by transferring the fixed point programming problems to solving a set of linear inequalities with the consideration of inconsistency, we use the feed forward nets and error back propagation algorithm to find the weights for the associative net. All the given patterns are made the fixed points of the obtained net with certain number of hidden neurons. Secondly, to meet the specification on the size of attraction basin, we pursue the model of Hopfield net with hidden structure even further. Through an orthogonalization process, the given patterns are made mutually orthogonal and the Hebbian rule is used for this set of orthogonal patterns to design an extended Hopfield net with hidden structure which guarantees all the given patterns to be its fixed points with the given size of attraction basin.

To overcome the problems of the inabilities of the Hebbian rule to guarantee any given patterns to be the fixed points with certain size of attraction basin, and at the same time, keep the number of the neurons used in the net minimum, a new method of designing Hopfield associative memory is developed as one of the important result of the thesis. Different from the Hebbian rule, this method explicitly introduce the size of the attraction basin as a design object and ends up with a Hopfield net with all the given patterns to be the fixed points with the largest possible *strict convergence ball*. From the examples in Section 3.4.4, it can be observed that the size of this largest *strict convergence ball* is increased compared to those of the net obtained by the Hebbian rule.

As an important model of neural networks which can be easily implemented by VLSI, cellular neural networks are playing roles in several areas of applications (see [15],[88] and [82] for some examples). Most of the present applications of cellular neural networks are in the domains of the image processing and pattern recognition. To find its place for cellular neural networks in the area of associative memory, part of this thesis is devoted to the design of such a neural associative memory and the study of the capacity issue as well as the constraints on the patterns to be stored.

Based on the Hopfield net, a new design rule has been developed and some modifications to the standard Hopfield structure have been made. These efforts end up with a better performance of the neural associative memories compared to the traditional Hopfield associative memories, but the result is still

not satisfactory. The main problem is: On the one hand, when we try to meet sharply all the requirements on the fixed points and the size of attraction basin, the net is probably stretched to an unreasonable size, i.e., with large number of hidden neurons. On the other hand, when we want to keep the number of the neurons relatively small, the size of the attraction basins will also be very small, especially when there are more patterns to be stored. It seems that we have to make our choice between the performance and the cost (in the sense of the number of neurons used) of the net. This is reasonable when the model is basically fixed (Hopfield net or its modified version).

Chapter 3 also explores the possibility of using the cellular neural nets as associative memories. The Hebbian rule is employed to set up the connection weights. We have analyzed the memory storage features, and have provided, among other things, a sufficient relation among the number of stored patterns, the size of the attracting basins and the minimal pattern distance.

As our worst case study shows that the cellular associative memory may pose serious limitations on the kind of patterns to be stored. To overcome the limitation, we are currently investigating a so-called cellular net with hidden neurons. The basic idea of the structure is to introduce the hidden neurons in the original net structure in a uniform way. At each basic local cell, we can assign (see Figure 3.19) arbitrary states to the hidden neurons so as to alleviate the stringent requirement posed on the given sequence of patterns.

Based on the discussions in Chapter 3, we have concluded that to further improve the performance (the number of the fixed points and the attraction basins surrounding each fixed point), the recurrent neural network model is not the only and best choice for the associative memories. In Chapter 4 we have developed a feedforward neural network model for the associations between binary patterns. Instead of using the dynamic convergent process to retrieve the stored memories like in the recurrent network case, the feedforward model uses a static mapping process to map the input patterns to the stored memories or associated patterns (a kind of desired outputs).

The obvious advantage in using a feedforward net for pattern association is that we only have to be concerned with algebraic rather than dynamic nature of the neural net. The latter is obviously much more intriguing and far less understood. It has been shown that with the feedforward net we have the perfect control over the size of the correction range. This is in contrast to the

feedback net for which the correction range will have to be made as the basin of attraction of a stable fixed-point, which is nearly impossible to be controlled by changing the weights and/or the thresholds only.

In terms of VLSI implementation, the feedforward net that we have discussed above also has greater simplicity than the feedback nets. In this case, subtle issues such as stability need not to be concerned in the circuit design. The drawbacks seem to be that the net structure is problem dependent, and the two-layer structure does not allow modular implementation of the net. Thus when the underlying matching problem is changed or goes to a higher dimension, the circuit will have to be redrawn to reflect the change and the existing circuit is not usable. Another disadvantage seems to be that the thresholds of the first layer depends upon the dimension of the input patterns. This is somehow undesirable as the higher input dimension will lead to higher thresholds. However, this latter difficulty can be avoided by the rearrangement of the algorithmic order of the network computations.

8.3 Nonlinear system identification and control

This part of the dissertation is very important in the whole research work, it represents an effort in applying artificial neural networks to a promising and practical field where traditional methods are not capable to deal with, i.e., nonlinear system identification (nonlinear signal modeling) and control. In this part we have developed a rule based neural controller for a benchmark nonlinear control problem, we have discussed the problem of control of a class of nonlinear systems based on the identification by feedforward multi-layer perceptron networks, and we have also proposed a novel weight updating algorithm and improved the center selection scheme for the radial basis function neural networks, which has been used for the multi-channel signal modeling and nonlinear system identifications.

Chapter 5 presented a heuristic neural control approach for solving nonlinear control problems. This approach is decision based, and has the task decomposition, the rule extraction and the neural net rule implementation as its essential elements. Via the specific control problem of the inverted pendulum, we have demonstrated how each of the three steps is actually considered and realized.

It may be of interest to compare this approach to other rule-based control approaches such as fuzzy control and expert control. It turns out that the rule-based neural control has following important advantages. First, it is able to implement dynamical decisions and other rules than just the static mapping actions. More precisely, in our feedback neural controller, the outcome of a sub-decision process or rule can be dependent on the previous outcome of its own. This dynamical decision process, lacking in both fuzzy and expert control approaches, strengthens considerably the decision ability, and sometimes proves to be indispensable for the decision control problems. In fact, it is the problem decomposition that necessitates the dynamical decision process. And decomposition is often considered to be a key to complex control problems.

As a fringe benefit, dynamic decision requires fewer rules than a purely static decision. This is easily understood as in the static case, time completely unfolds resulting in the need to cover all the time aspects of the decision.

Secondly, once constructed, the rule-based neural controller is a deterministic nonlinear dynamical system (with local feedback only), thus allowing the analysis to be carried out for performance evaluation. This is in contrast to both fuzzy and expert controls, where their final linguistic constructs prevent an analytic evaluation.

Finally, because of its parallel nature the rule-based neural controller is computationally advantageous over fuzzy and expert control. The computational advantage also follows from the reduced number of rules as a result of dynamical decision process. Like all the other rule-based approaches, the rule-based neural controller is inherently non-analytic and imprecise. The ambiguity follows from the heuristic and subjective nature of the problem decomposition and rule extraction. Moreover, the approach does not intend to solve all complex nonlinear control problems. What it really attempts to solve is the nonlinear control problems that are less well-defined, but nevertheless seem intuitively solvable.

Chapter 6 and 7 systematically presented the modeling of multi-channel signals and the identification of nonlinear dynamical systems by the feedforward multi-layer perceptron networks and radial basis function networks as well as nonlinear system control based on identification. For the model of multilayer feedforward neural networks we proposed an off-line and an on-line identification scheme for a rather broad class of nonlinear dynamical

systems. Based on the off-line identification, an optimization based one-step-ahead predictive control scheme is discussed. This adaptive control scheme, which combines the further on-line identification of the system with the control process, is proven to be effective and is especially preferable when the plant to be controlled is possibly changing its parameters during the identification and control process.

For the RBF neural network model, we have developed a discrete-time nonlinear signal modeling scheme based on this type of neural nets. The key motivation is to carry over the essence of the linear ARMA modeling to the nonlinear case with the help of RBF neural nets. In this particular framework, the modeling problem is decomposed into two separate sub-problems of determining the weights and the structural parameters. The theory of spatial sampling and the Fourier analysis helps to find the structural parameters in the light of function approximation. Once the structure is known, a stable recursive algorithm can be readily used to find the weights of the nets. The convergence of the algorithm is relatively easy to establish compared to other types of neural nets where convergence often comes as an accidental by-product of a prolonged training process.

Compared to many existing modeling techniques, such as the back propagation net based techniques, the present one has the clear advantage of accuracy-efficiency tradeoff, with the typical training cycle of a few thousands to reach to within the mean-square modeling error in the order of 1%. Moreover, compared to other types of RBF net based modeling techniques, each of the basic steps of the present modeling scheme is backed by theoretical considerations rather than pure heuristic considerations.

The use of RBF neural nets is often hindered by the perception that a RBF net would necessarily involve a great deal more neurons (thus more weights) than other kind of neural nets (*e.g.*, a Back-propagation net) for exactly the same approximation problem. Our experiences with the scheme indicate, however, that this needs not be the case, especially in terms of the number of weights for achieving the same accuracy. The examples given in the simulation section of Chapter 7 have partially illustrated this point.

One should keep in mind that our modeling scheme is only applicable under many assumptions, including the existence of a compact operating region. It is after all an approximating modeling scheme that will never be able to generate

an exact model for a given pair of input-output sequences. *Approximate modeling* is actually a key starting point for the development of the scheme. It is not yet clear if there exists a constructive solution for the problem of *exact modeling* on a compact set or beyond.

8.4 General conclusions

Hopfield network and feedforward networks (MLP and RBF) are currently two most important types of networks. Their structures are quite simple with either feedforward connections or full connections. Many learning algorithms for these two types of networks were proposed and studied extensively in the past years and progresses have been made in improving the efficiency, convergence speed, applicability and robustness.

Starting in the beginning of 1980's, the work on Hopfield associative memories had attracted many researchers in different disciplines, especially in physics, computer science and electrical engineering. The objectives of these works have been focused on the stability of the dynamics, weight setup rules, fixed point programming, spurious states, storage capacity and the sizes of attraction basins. In the course of these studies, among other things, different learning algorithms were derived for the standard Hopfield network structure, modified Hopfield network was discussed to come up with certain requirements on the memories and a feedforward model of associative memories was proposed. Compared with the work in other fields of artificial neural networks, the works on neural associative memories are mostly on theoretical aspects and the results obtained are still a bit far from the practical applications, which will also depend on the availability of large scale hardware implementation of neural networks.

The latter part of this dissertation has been devoted to the problem of nonlinear system identification and control by feedforward neural networks. Considering the approximation, learning and self adaptation abilities of this type of networks, it seems that neural networks have a lot of things to do here, especially when there are not many good solutions to the nonlinear problems in traditional approaches. Future works in this area can be directed to the selection of the best network model among several types of feedforward networks, or determine the suitable type of networks for different classes of

nonlinear problems. RBF networks have been studied in this dissertation and is believed to be one of the most appropriate ones for this type of problems. Further work on RBF includes the improvement to the learning algorithm for the weights and finding more efficient center selection method.

Bibliography

[1] E. Aarts and J. Korst (1989), *Simulated Annealing and Boltzmann Machines*, Jone Wiley & Sons, New York.

[2] C. W. Anderson (1989), "Learning to control an inverted pendulum using neural networks," *IEEE Control Systems Magazine*, Vol.9, pp.31-37.

[3] K. J. Aström and B. Wittenmark (1989), *Adaptive Control*, Reading, MA, Addison-Wesley.

[4] R.H. Baran (1989), "A collective computation approach to automatic target recognition", *Proc. IJCNN 1989*, Washington D.C., Vol.1, pp.39-44.

[5] A. G. Barto, R. S. Sutton and C. W. Anderson (1983), "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Trans. System, Man and Cybernetics*, Vol.SMC-13, pp.834-846.

[6] E.B. Baum (1988), "On the capabilities of multilayer perceptrons," *J. Complexity*, Vol.4, pp.193-215.

[7] M.L. Brady, R. Raghavan and J. Slawny (1989), "Back propagation fails to separate where perceptrons succeed," *IEEE Trans. Circ. Syst.*, Vol.CAS-36, pp.665-674.

[8] D. S. Broomhead and D. Lowe (1988), "Multivariable functional interpolation and adaptive networks," *Complex Systems*, Vol.2, pp.321-355.

[9] J. Bruck and J.W. Goodman (1988), "A generalized convergence theorem for neural networks," *IEEE Trans. Inform. Theory*, Vol.34, pp.1089-1092.

[10] J. Bruck and J. Sanz (1988), "A study on neural networks," *Int. J. Intelli. Syst.*, Vol.3, pp59-75.

[11] R. H. Cannon (1967), *Dynamics of Physical Systems*. New York: McGraw-Hill.

[12] G.A. Carpenter and S. Grossberg (1987), "A massively parallel architecture for a self-organizing neural pattern recognition machine," *computer Vision, Graphics, and Image Processing*, Vol.37, pp.54-115.

[13] G.A. Carpenter and S. Grossberg (1987), " ART 2: Self-organization of stable category recognition codes for analog input patterns," *Applied Optics*, Vol.26, pp.4919-4930.

[14] L.O. Chua and L. Yang (1988), "Cellular neural networks: Theory," *IEEE Trans. Circ. Syst.*, Vol.CAS-35, pp.1257-1272.

[15] L.O. Chua and L. Yang (1988), "Cellular neural networks: Application," *IEEE Trans. Circ. Syst.*, Vol.CAS-35, pp.1273-1290.

[16] D. W. Clarke, C. Mohtad and P. S. Tuffs (1987), "Generalized predictive control — Part. I, The basic algorithm," *Automatica*, Vol.23, pp.137-148.

[17] A. Dembo (1989), "On the capacity of associative memories with linear threshold functions," *IEEE Trans. Inform. Theory*, Vol.35, No.4, pp.709-720.

[18] J.S. Denker (1986), "Neural network models of learning and adaptation," *Physica*, Vol. 22D, pp.216-232.

[19] J.A. Farrell and A.N. Michel (1990), "A synthesis procedure for Hopfield's continuous-time associative memory", *IEEE Trans. Circuits Syst.*, Vol.37, pp.877-884.

[20] P. Floreen and P. Orponen (1989), "Counting stable states and sizes of attraction domains in Hopfield nets is hard," *Proc. IJCNN* 1989, Washington D.C., Vol.1, pp.395-399.

[21] K. Fu (1970), "Learning control systems — Review and outlook," *IEEE Trans. Autom. Control*, Vol.AC-15, pp.210-221.

[22] K. I. Funahashi (1989), "On the approximate realization of continuous mappings by neural networks," *Neural Net.*, Vol.2, pp.183-192.

[23] S. Grossberg (1988), "Nonlinear neural networks: principles, mechanisms, and architectures," *Neural Net.*, Vol.1, pp.17-61.

[24] S. Grossberg (1976), "Adaptive pattern classification and universal recording, II: Feedback, expectation, olfaction, and illusions," *Biological Cybernetics*, Vol.23, pp.187-202.

[25] A. Guez and J. Selinsky (1988), "A neuromorphic controller with a human teacher," *Proc. IEEE Intern. Conf. on Neural Net.*, San Diego, Vol.2, pp.595-602.

[26] J. Hao, S. Tan and J. Vandewalle (1990), " A geometrical approach to the structural synthesis of multilayer perceptron neural networks," *Proceedings of International Neural Network Conference (INNC-90, Paris)*, Vol.2, pp.881-885.

[27] J. Hao, S. Tan and J. Vandewalle (1991), "A new approach to the design of discrete Hopfield associative memory," *Proc. IJCNN'91*, Singapore, Vol.2, pp.1705-1710.

[28] J. Hao, S. Tan and J. Vandewalle (1992), "Bipolar pattern association using a two-layer feedforward neural network," *Proc. IJCNN'92*, Beijing, Vol.2, pp.65-71.

[29] J. Hao and J. Vandewalle (1992), "A new model of neural associative memories," *Proc. IJCNN'92*, Baltimore, Vol.2, pp.166-171.

[30] J. Hao, S. Tan and J. Vandewalle (1993), " Neural network implementation of heuristic rule based nonlinear control," *Proc. Asia-Pacific*

Workshop on Advances in Motion Control, Singapore, 1993, pp.218-223.

[31] J. Hao and J. Vandewalle (1993), "Inverse identification for control of nonlinear dynamical systems by back propagation neural networks," *Proc. World Congress on Neural Networks '93*, Portland, Vol.3, pp.317-320.

[32] J. Hao, S. Tan and J. Vandewalle (1993), "One step ahead predictive control of nonlinear systems by neural networks," *Proc. IJCNN'93*, Nagoya, Vol.3, pp.2761-2764.

[33] J. Hao, S. Tan and J. Vandewalle (1993), "A rule-based neural controller for inverted pendulum system," *International Journal of Neural Systems*, Vol.4, No.1, pp.55-64.

[34] J. Hao, S. Tan and J. Vandewalle (1993), "Bipolar pattern matching using multilayer perceptron neural networks," *IEEE Trans. Circuits Syst.*, Part I, Vol.40, No.12, pp.943-946.

[35] J. Hao, S. Tan and J. Vandewalle (1994), "A new approach to the design of discrete Hopfield associative memory," *Journal of Artificial Neural Networks*, Vol.1, No.2, pp.247-266.

[36] J. Hao and J. Vandewalle (1994), "A new model of neural associative memories," *International Journal of Neural Systems*, Vol.5, No.1, pp.39-47.

[37] J. Hao, S. Tan and J. Vandewalle (1994), "Predictive control of nonlinear systems based on identification by backpropagation networks," *International Journal of Neural Systems*, in press.

[38] G.E. Hinton and J.A. Anderson (1981), Eds., *Parallel Models of Associative Memory*, New Jersey: Lawrence Erlbaum Assoc.

[39] M.W. Hirsch (1989), "Convergent activation dynamics in continuous time networks," *Neural Net.*, Vol.2, pp.331-349.

[40] L. Hitz and B.D.O. Anderson (1969), "Discrete positive-real functions and their application to system stability," *Proc. IEE*, pp. 153-155.

[41] J.E. Hopcroft and R.L. Mattson (1965), "Synthesis of minimal threshold logic networks," *IEEE Trans. Elec. Computers*, Vol.EC-6, pp.552-560.

[42] J.J. Hopfield (1982), "Neural networks and physical systems with emergent collective computational abilities," *Proc. Nat. Acad. Sci. USA*, Vol.79, pp.2554-2558.

[43] J.J. Hopfield (1984), "Neurons with graded response have collective computations properties like those of two-state neurons," *Proc. Nat. Acad. Sci. USA*, Vol.79, pp.2554-2558.

[44] K. Hornik, M. Stinchcombe and H. White (1989), "Multilayer feedforward networks are universal approximators," *Neural Net.*, Vol.2, pp.359-366.

[45] W.Y. Huang and R.P. Lippmann (1988), "Neural net and traditional classifiers," *Neural Information Processing Systems*, Ed. D.Z. Anderson, pp.387-396.

[46] K. J. Hunt, D. Sbarbaro, R. Zbikowski and P. J. Gawthrop (1992), "Neural networks for control systems – A survey," *Automatica*, Vol. 28, pp. 1083-1112.

[47] K. Hwang and F.A. Briggs (1984), *Computer Architecture and Parallel Processing*, New York: McGraw-Hill Book Company.

[48] A. Isidori (1989), *Nonlinear Control Systems*, Berlin, Springer-Verlag.

[49] M. Kam, J. Chow and R. Fischl (1990), "Design of the fully connected binary network via linear programming," *Proc. ISCAS* 1990, Vol.2, pp.1094-1097.

[50] M. Kawato, Y. Uno, M. Isobe and R. Suzuki (1988), "Hierarchical neural network model for voluntary movement with application to robotics," *IEEE Control Systems Magazine*, Vol.8, pp.84-98.

[51] T. Kohonen (1984), *Self-Organization and Associative Memory*, New York: Springer-Verlag.

[52] B. Kosko (1988), "Bidirectional associative memories", *IEEE Trans. Syst. Man Cyber.*, Vol.18, pp.49-60.

[53] S. Lee and R.M. Kil (1988), "Multilayer feedforward potential function network," *Proc. IEEE Intern. Conf. Neural Net.*, Vol. 1, pp.161-171.

[54] S. Lee and R.M. Kil (1991), "A Gaussian potential function network with hierarchical self-organizing learning," *Neural Net.*, Vol. 4, pp.207-224.

[55] Y. H. Lin and K. S. Narendra (1980), "A new error model for adaptive systems," *IEEE Trans. Autom. Control*, Vol. AC-25, pp. 585-587.

[56] R.P. Lippmann (1988), "An introduction to computing with neural nets," *IEEE ASSP Magazine*, April, pp.4-22.

[57] J. McMlelland and David Rumelhart (1986), *Parallel Distributed Processing*, Cambridge: MIT Press and the PDP Research Group.

[58] W.S. McCulloch and W.H. Pitts (1943), "A logical calculus of the ideas imminent in nervous activity," *Bull. Math. Biophy.*, Vol.5, pp.115-133.

[59] R.J. McEliece, E.C. Posner, E.R. Rodemich and S.S. Venkatesh (1987), "The capacity of the Hopfield associative memory", *IEEE Trans. Inform. Theory*, Vol.33, No.4, pp.461-482.

[60] C.A. Micchelli (1986), "Interpolation of scattered data: Distance matrices and conditionally positive definite functions," *Constructive Approximation*, Vol.2, pp.11-22.

[61] A.N. Michel and J.A. Farrell (1990), "Associative memories via artificial neural networks," *IEEE Contr. Syst. Magaz.*, April, pp.6-17.

[62] M.L. Minsky and S.A. Papert (1988), *Perceptron: An Introduction to Computational Geometry*, expanded ed., MIT Press, MA.

[63] G. Mirchandani and W. Cao (1989), "On hidden nodes for neural nets," *IEEE Trans. Circ. Syst.*, Vol.CAS-36, pp.661-664.

[64] J. Moody and C. Darken (1989), "Fast learning in networks of locally-tuned processing units," *Neural Computation*, Vol. 1, pp.281-294.

[65] Y.S. Mostafa and J.S. Jacques (1985), "Information capacity of the Hopfield model," *IEEE Trans. Inform. Theory*, Vol.31, No.4, pp.461-464.

[66] K. S. Narendra and A. M. Annaswamy (1989), *Stable Adaptive Control*, Prentice-Hall, Englewood Cliffs, NJ.

[67] K. S. Narendra and A. M. Annaswamy (1989), *Stable Adaptive Systems*, Englewood Cliffs, NJ, Prentice-Hall.

[68] K. S. Narendra and K. Parthasarathy (1990), "Identification and control of dynamical systems using neural networks," *IEEE Trans. on Neural Networks*, Vol.1, pp.4-27.

[69] D. Nguyen and B. Widrow (1989), "The truck backer-upper: an example of self-learning in neural networks," *Proc. IEEE Intern. Joint Conf. on Neural Net.*, Washington DC, pp.357-363.

[70] H. Nijmeijer and A.J. van der Schaft (1990), *Nonlinear Dynamical Control Systems*, New York, NY, springer-Verlag,

[71] Z. J. Nikolic and K. S. Fu (1966), "An algorithm for learning without external supervision and its application to learning control," *IEEE Trans. Autom. Control*, Vol.AC-11, pp.414-422.

[72] N.J. Nilsson (1965), *Learning Machines*, McGraw-Hill Book Company, New York.

[73] M. Niranjan and F. Fallside (1990), "Neural networks and radial basis functions for classifying static speech patterns," *Computer Speech and Language*, Vol.4, pp.275-289.

[74] J. Park and I. W. Sandberg (1990), "Universal approximation using radial-basis-function networks," *Neural Computation*, Vol.3, pp.246-257.

[75] J. Platt (1991), "A resource-allocation network for functional approximation," *Neural Computation*, Vol. 3, pp.213-225.

[76] M.M. Polycarpou and P.A. Ioannou (1991), "Identification and control of nonlinear systems using neural network models: design and stability analysis," Technical Report 91-09-01, Dept. Elec. Eng., Univ. of Southern Cal.

[77] D. Psaltis, A. Sideris and A. Yamamura (1988), "A multilayered neural network controller," *IEEE Control Systems Magazine*, Vol.8, pp.17-21.

[78] U. Ramacher and M. Wesseling (1989),"A geometrical approach to neural network design," *Proc. Intern. Joint Conf. Neural Net.*, Vol.2, pp.147-153.

[79] S. Renals and R. Rohwer (1989), "Phoneme classification experiment using radial basis functions," *Proc. Intern. Joint Conf. Neural Net.*, Vol. 1, pp. 461-467.

[80] M. Röscheisen *et.al* (1991), "Incorporating prior knowledge in parsimonious networks of locally-tuned units," *Siemens Technical Report TR-FKI-155-91*.

[81] F. Rosenblatt (1962), *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanism*, Spartan Books, Washington.

[82] T. Roska, T. Boros, P. Thiran and L.O. Chua (1990), "Detecting simple motion using cellular neural networks," *Proc. IEEE CNNA-90*, Budapest, pp.127-138.

[83] P. Ruján (1989),"A geometric approach to learning in neural networks," *Proc. Intern. Joint Conf. Neural Net.*, Vol.2, pp.105-109.

[84] D.E. Rumelhart, G.E. Hinton and R.J. Willians (1986), "Learning internal representations by error propagation," *Parallel Distributed Processing*, Vol.1 *Foundations*, (MIT Press, Cambridge) pp.318-362.

[85] R. M. Sanner and J. E. Slotine (1992), "Gaussian networks for direct adaptive control," *IEEE Trans. Neural Net.*, Vol. 3, pp.837-863.

[86] G. N. Saridis (1981), "Application of pattern recognition methods to control systems," *IEEE Trans. Autom. Control*, Vol.AC-26, pp.638-645.

[87] S. S. Sastry and A. Isidori (1989), "Adaptive control of linearizable system," *IEEE Trans. Aut. Control,* Vol-AC34, pp.1123-1131.

[88] G. Seiler (1990), "Small object counting with cellular neural networks," *Proc. IEEE CNNA-90,* Budapest, pp.114-123.

[89] M. Sugeno (1985), "An introductory survey of fuzzy control," *Inform. Sci.,* Vol.36, pp.59-83.

[90] S. Tan, J. Hao and J. Vandewalle (1990), "Cellular neural networks as a model of associative memories," *Proceedings of 1990 IEEE International Workshop on Cellular Neural Networks and Their Applications (CNNA-90, Hungary),* pp.26-35.

[91] S. Tan, B. De Moor, J. Hao and J. Vandewalle (1990),"Complex nonlinear control by simple neural decision networks," *ESAT-SISTA Internal Report,* No.90-43I.

[92] S. Tan, J. Hao and J. Vandewalle (1991), "Determination of weights for Hopfield associative memory by error backpropagation," *Proc. IEEE International Symposium on Circuits and Systems '91,* Singapore, pp.2491-2494.

[93] S. Tan, J. Hao and J. Vandewalle (1992), "Pattern storage and Hopfield neural associative memory with hidden structure," *International Journal of Neural Systems,* Vol.3, No.3, pp.315-322.

[94] S. Tan, J. Hao and J. Vandewalle (1993), "Nonlinear systems identification using RBF neural networks", *Proc. IJCNN'93,* Nagoya, Vol.2, pp.1833-1836.

[95] S.Tan, J.Hao and J.Vandewalle (1994), "Identification of nonlinear systems by RBF neural networks," *Proc. ICANN'94,* Sorrento, Vol.2, pp.1203-1206.

[96] S.Tan, J.Hao and J.Vandewalle (1994), "Identification of nonlinear discrete-time multivariable dynamical systems by RBF neural networks," *Proc. WCCI-ICNN'94,* Orlando, Vol.5, pp.3250-3255.

[97] S. Tan, J. Hao and J. Vandewalle (1994), "Efficient identification of RBF neural net models for nonlinear discrete-time multivariable dynamical systems," *Neurocomputing,* in press.

[98] S. Tan, J. Hao and J. Vandewalle (1994), "Stable and efficient neural network modeling of discrete multi-channel signals," *IEEE Trans. Circuits Syst,* in press.

[99] D. G. Taylor, P. V. Kokotovic, R. Marino and I. Kanellakopoulos (1989), "Adaptive regulation of nonlinear systems with unmodelled dynamics," *IEEE Trans. Aut. Control,* Vo.AC-34, pp.405-412.

[100] V. V. Tolat and B. Widrow (1988), "An adaptive broom balancer with visual inputs," *Proc. IEEE Intern. Conf. on Neural Net.,* San Diego, Vol.2, pp.641-647.

[101] L. Vandenberghe, S. Tan and J. Vandewalle (1989), "Cellular neural networks: Dynamic properties and adaptive learning algorithm," *Proceedings of EURASIP workshop on Neural Networks,* pp.141-150, lecture notes in Computer Science, L.B. Almeida, C.J. Wellekens Ed. Springer-Verlag.

[102] S.S. Venkatesh and D. Psaltis (1989), "Linear and logarithmic capacities in associative neural networks," *IEEE Trans. Infor. Theory,* Vol.IT-35, pp.459-468.

[103] B. Widrow (1962), "Reliable, trainable networks for computing and control," *Aerospace Engineering,* September, pp.78-123.

[104] B. Widrow (1987), "The original adaptive neural net broom-balancer," *Int. Symp. Circuits and Syst.,* pp.351-357.

[105] B. Widrow and M.A. Lehr (1990), "30 years of adaptive neural networks: Perceptron, madaline, and backpropagation," *Proc. IEEE,* Vol.78, pp.1415-1441.

[106] R.O.Winder (1963), "Bounds on threshold gate realizability," *IRE Trans. Electron. Comp.,* Vol.EC-12, pp.561-564.

[107] J.M. Zurada (1992), *Artificial Neural Systems*, West Publishing Company, St. Paul, MN.