Cambridge University Engineering Department
Trumpington Street
Cambridge CB2 1PZ

On the Theory of Generalization and Self-Structuring in Linearly Weighted Connectionist Networks

Sean B. Holden[1]

CUED/F-INFENG/TR.161

January 25, 1994

[1] email: sbh@eng.cam.ac.uk

On the Theory of Generalization and Self-Structuring in Linearly Weighted Connectionist Networks

by
Sean Barry Holden

Corpus Christi College

A dissertation submitted to the
University of Cambridge for the Degree
of Doctor of Philosophy

Department of Engineering September 26, 1993

To my parents

Declaration

The research described in this dissertation was carried out by the author between October 1989 and September 1993. Except as indicated in the text, the contents are entirely original and are not the result of work done in collaboration. No part of this dissertation has been submitted to any other University. The length of the dissertation is less than 55,800 words in total.

Sean B. Holden

Acknowledgements

Thanks are due, first and foremost, to my supervisor Dr. Peter Rayner for his advice and support during the last few years. I also thank those who answered my unsolicited emails and phone calls asking for unpublished work.

One such person was Dr. Martin Anthony, who I also thank, along with Dr. Steve Renals, for proofreading the dissertation. Thanks for proofreading various parts of the dissertation are also due to Ray Auchterlounie, Robin Morris, Simon Godsill, Dr. Anil Kokaram, Dr. Malcolm MacLeod and Dr. Mahesan Niranjan.

I acknowledge the help of the Science and Engineering Research Council, Corpus Christi College, and Cambridge University Engineering Department, who provided funding for maintenance and for attending several conferences.

I thank my family, for providing enormous and greatly needed support during my time in Cambridge and, for that matter, all the time before that as well. I have made many friends in Cambridge all of whom have been tremendous fun, and thanks are due to them all. In particular, I thank the past and present members of Corpus Christi College Boat Club.

Keywords

The following keywords may be useful for indexing purposes:

Connectionist networks; generalization; probably approximately correct (PAC) learning theory; self-structuring; radial basis function networks; Volterra networks; regularization networks; modified Kanerva models; Vapnik-Chervonenkis (VC) dimension; growth function; polynomial discriminant functions; linearly weighted connectionist networks; perceptrons; multilayer perceptrons; hidden layer back-propagation; computational learning theory; capacity; interpolation; least squares; weight decay; least mean squares (LMS) algorithm; singular value decomposition (SVD); minimum norm.

Summary

The study of connectionist networks has often been criticized for an overall lack of rigour, and for being based on excessively ad hoc techniques. Even though connectionist networks have now been the subject of several decades of study, the available body of research is characterized by the existence of a significant body of experimental results, and a large number of different techniques, with relatively little supporting, explanatory theory. This dissertation addresses the theory of *generalization performance* and *architecture selection* for a specific class of connectionist networks; a subsidiary aim is to compare these networks with the well-known class of multilayer perceptrons.

After discussing in general terms the motivation for our study, we introduce and review the class of networks of interest, which we call Φ-*networks*, along with the relevant supervised training algorithms. In particular, we argue that Φ-networks can in general be trained significantly faster than multilayer perceptrons, and we demonstrate that many standard networks are specific examples of Φ-networks.

Chapters 3, 4 and 5 consider generalization performance by presenting an analysis based on tools from computational learning theory. In chapter 3 we introduce and review the theoretical apparatus required, which is drawn from *Probably Approximately Correct (PAC) learning theory*. In chapter 4 we investigate the *growth function* and *VC dimension* for general and specific Φ-networks, obtaining several new results. We also introduce a technique which allows us to use the relevant PAC learning formalism to gain some insight into the effect of training algorithms which adapt architecture as well as weights (we call these *self-structuring training algorithms*). We then use our results to provide a theoretical explanation for the observation that Φ-networks can in practice require a relatively large number of weights when compared with multilayer perceptrons. In chapter 5 we derive new necessary and sufficient conditions on the number of training examples required when training a Φ-network such that we can expect a particular generalization performance. We compare our results with those derived elsewhere for feedforward networks of Linear Threshold Elements, and we extend one of our results to take into account the effect of using a self-structuring training algorithm.

In chapter 6 we consider in detail the problem of designing a good self-structuring training algorithm for Φ-networks. We discuss the best way in which to define an optimum architecture, and we then use various ideas from linear algebra to derive an algorithm, which we test experimentally. Our initial analysis allows us to show that the well-known *weight decay* approach to self-structuring is not guaranteed to provide a network which has an architecture close to the optimum one. We also extend our theoretical work in order to provide a basis for the derivation of an improved version of our algorithm.

Finally, chapter 7 provides conclusions and suggestions for future research.

Contents

List of Figures

List of Tables

Glossary

Abbreviations

The following abbreviations are used in this dissertation.

CMAC	Cerebellar Model Articulation Controller
HLBP	Hidden Layer Back-Propagation
LDF	Linear Discriminant Function
LTE	Linear Threshold Element
MKM	Modified Kanerva Model
MLP	Multilayer Perceptron
PAC	Probably Approximately Correct
PDF	Polynomial Discriminant Function
RBF	Radial Basis Function
RBFN	Radial Basis Function Network
RN	Regularization Network
SVD	Singular Value Decomposition
VC	Vapnik-Chervonenkis

Mathematical conventions

Matrices and vectors are printed in bold text; matrices are printed in upper case and vectors in lower case. All vectors are column vectors. The notation $\mathbf{A}^T$ or $\mathbf{a}^T$ is used to denote the transpose of a matrix or vector and $\mathbf{A}^+$ denotes the Moore-Penrose pseudoinverse. We say that a function $f(x)$ is $O(g(x))$ if there are constants a and b, both greater than 0, such that for all x, $f(x) \leq ag(x) + b$. Similarly, we say that a function $f(x)$ is $\Omega(g(x))$ if there are constants a and b, both greater than 0, such that for all x, $f(x) \geq ag(x) + b$. The following is a summary of the main mathematical notation used in the dissertation. The list is not exhaustive; it contains only the most important and frequently used symbols. Other notation is defined in full in the text.

$\|\mathbf{v}\|$	The Euclidean norm of a vector $\mathbf{v}$.
2^S	The set of all subsets of some set S.
$\triangle_H(i)$	The growth function of H.
$\mathbb{B}$	The set $\{+1, -1\}$.
$\mathrm{cs}(\mathbf{A})$	The column space of a matrix $\mathbf{A}$.
$\mathcal{C}$	The set of class indices.
C^i	The set of i-times continuously differentiable mappings on a specified interval.
C^i_j	The binomial coefficient, $C^i_j = i!/j!(i-j)!$.
$\dim(V)$	The dimension of a vector space V.
ϵ	The actual error of a network, defined as the probability that it classifies a test input incorrectly after training.
$\epsilon_i(\mathbf{w})$	The error of a network for the ith training example.
e	The base of the natural logarithm (the Euler number).
E	The number of edges in the graph G for a feedforward network.
$E[.]$	The expected value of a random variable.

$f_{\mathbf{w}}$	The function computed by a network having weight vector $\mathbf{w}$.
$\mathcal{F}$	The class of functions computed by a general network.
$\mathcal{F}_n$	The class of linear threshold functions having n inputs.
$\mathcal{F}_n^{\Phi}$	The class of functions computed by a Φ-network having n inputs and the set Φ of basis functions.
G	The graph describing the architecture of a feedforward network.
k	The number of examples in a sequence of training examples.
k_0	Used to denote bounds on sufficient numbers of training examples.
ln	The natural logarithm.
$\log_i$	The logarithm to base i.
m	The number of basis functions used by a Φ-network.
n	The number of inputs to a network.
N	The number of computation nodes in a feedforward network.
$\mathcal{N}$	The 5-tuple describing a Φ-network.
$\mathcal{N}'$	The 5-tuple describing the Φ-network searched for using self-structuring.
$N(\mathbf{A})$	The nullspace of a matrix $\mathbf{A}$.
$N(i,j)$	The number of ways of choosing a set of unconstrained weights for an l-restriction. $N(i,j) = C_{i-1}^{j-1}$.
o_i	The ith desired output in a sequence of training examples.
$\mathbf{o}$	Vector containing the k desired outputs o_i.
$\mathbf{p}$	The vector equal to $\mathbf{P}^T\mathbf{o}$.
P	A probability distribution on X.
$\mathbf{P}$	Matrix containing as rows the transposes of the extensions of the k input vectors in a sequence of training examples.
$\mathcal{P}$	The probability that a trained network has an actual error greater than a specified value.
P'	A probability distribution on $X \times \mathbb{B}$.
$\mathbb{P}_d(\mathbb{R}^n)$	The set of strictly conditionally positive definite functions of order d.
Pr[.]	The probability of an event.
ϕ	Radial basis function.
Φ	Set of basis functions used by a Φ-network.
$\boldsymbol{\varphi}$	The function used to construct an extended vector.
ϕ_i	The ith basis function used by a Φ-network.
Φ_i	A subset of Φ.
$\pi_{d-1}(\mathbb{R}^n)$	The linear space of algebraic polynomials from $\mathbb{R}^n$ to $\mathbb{R}$ of degree at most $(d-1)$.
Ψ	The function specifying the number of weights in a polynomial discriminant function.
r	Used to denote the rank of a matrix.
$R(\mathbf{A})$	The range of a matrix $\mathbf{A}$.
$R(\mathbf{w})$	Function used to penalize weight vectors having many non-zero weights.
$\mathbf{R}$	The matrix equal to $\mathbf{P}^T\mathbf{P}$.
$\mathbb{R}$	The set of real numbers.
$\mathbb{R}^+$	The set $\{x \mid x \in \mathbb{R} \text{ and } x > 0\}$.
$\mathbb{R}_0^+$	The set $\mathbb{R}^+ \cup 0$.
$\mathbb{R}^i$	The set of all real vectors with i elements.

$\mathbb{R}^{i\times j}$	The set of i-by-j matrices with real-valued elements.
$\mathcal{R}^l(.)$	The l-restriction of a Φ-network.
$\mathrm{rs}(\mathbf{A})$	The row space of a matrix $\mathbf{A}$.
$\rho(x)$	The function equal to $+1$ if $x \geq 0$ and -1 otherwise.
σ	The activation function used by a multilayer perceptron. Also used to denote the width of a radial basis function.
σ_i	The ith singular value of a matrix. Also used to denote the width of a radial basis function.
s_1, s_2	Used to denote two standard sigmoid functions.
S	The solution space for a least squares problem.
S'	The augmented solution space.
S_k	A set of k elements of an environment.
$\mathrm{span}\{.\}$	The set of all linear combinations of the specified set of vectors.
T_k	A sequence of k training examples.
$\mathcal{V}(H)$	The Vapnik-Chervonenkis dimension of H.
$\mathbf{w}$	$\mathbf{w}^T = [\ w_0 \quad w_1 \quad \cdots \quad w_m\ ]$ is the weight vector of a connectionist network.
$\mathbf{w}_{\mathrm{min}}$	The minimum norm weight vector.
$\mathbf{w}_{\mathrm{opt}}$	Any optimum solution to a least squares problem.
W	The number of weights used by a network.
W'	Number of unconstrained weights. Used when modelling self-structuring using an l-restriction.
$\xi(\mathbf{w})$	The error on the training examples of a network having weight vector $\mathbf{w}$.
$\mathbf{x}$	$\mathbf{x}^T = [\ x_1 \quad x_2 \quad \cdots \quad x_n\ ]$ is the vector of n inputs to a network.
$\tilde{\mathbf{x}}$	$\tilde{\mathbf{x}}$ is the extended form of an input vector $\mathbf{x}$.
$\mathbf{x}_i$	The ith input in a sequence of training examples.
X	The environment of a network. Corresponds to the input space of the network.
$\mathbf{y}_i$	The vector of parameters associated with the ith basis function in a Φ-network with parameterized basis functions.

Chapter 1

Introduction

1.1 The Scope of this Dissertation

A great deal has been written in the last few decades on the subject of *connectionist networks* and their potential applications as pattern classifiers, associative memories, predictors *etc.* A great deal has been said about their apparent ability to learn from examples, deal sensibly with badly defined problems, perform generalization, and be tolerant to faults, but there has tended to be relatively little work done in an attempt to gain a *fundamental* understanding of these properties — connectionist network research, and the discipline of connectionist network design, have often been criticized for being too ad hoc, and even after several decades of study much connectionist network design is based on a process of trial and error.

Two major reasons can be identified which have led to this state of affairs. Firstly, researchers have tended to study their networks with little regard for alternative and generally better understood strategies for approaching similar problems; as an obvious example, we note that in most connectionist network research relatively little regard is given to whether better results could be obtained using standard statistical methods. Secondly, the field is characterized by the existence of a large body of experimental results with relatively little accompanying work on the production of good explanatory theories. A prime example here is the investigation of the property of generalization, and we expand on this issue below.

The comparison of connectionist networks with alternative techniques, and the theoretical analysis of the properties of these networks which are perceived to be of interest, are thus important areas for research, and indeed in recent years a welcome development in connectionist network research has been the increase in the willingness of researchers to address these issues. Omohundro [1] for example provides an extensive study of how standard computational algorithms can be used to obtain results similar to those produced by connectionist networks. Similarly, Ripley [2] and others have compared connectionist networks with standard statistical techniques. Other examples are available, however this is not the major concern of this dissertation and so we do not mention them here. The contribution of this dissertation concerns the second of the two areas suggested: the theoretical analysis of connectionist networks.

In this dissertation we provide a rigorous analysis of two specific aspects of a specific class of feedforward connectionist networks, trained using supervised learning. The actual class with which we shall be interested was chosen for quite specific reasons. Originally we found these networks interesting because they appeared to provide excellent practical results while being significantly easier to train than the usual feedforward network of choice: the multilayer perceptron (MLP) trained using hidden layer back-propagation. What increased our interest was the fact that the class in general appears to be considerably less popular — it is of course interesting to ask why!

These networks are also interesting because they are quite analytically tractable, and in this sense our inspiration is similar to that of Minsky and Papert [3]. These authors, in approaching their celebrated work, had very specific reasons for studying the class of perceptrons: they were simple enough to be analytically tractable, and flexible enough to be interesting. We study a particular class of networks — similar in some ways to perceptrons which are in fact a subclass — for two reasons. Firstly, they combine flexibility with analytical tractability in the same manner as perceptrons. Secondly, they have proved highly successful in practical applications — in particular, as we shall see, they have significant advantages over many alternative networks, in particular the MLP, in terms of training time while often providing comparable performance.

We will be interested in two specific aspects of the design of these networks: generalization performance and the selection of a good network architecture. These aspects, and the reasons for studying them, are explained below. In both cases we draw on work not specific to the study of connectionist networks. In the former case we use ideas from computational learning theory, specifically from Probably Approximately Correct (PAC) learning, and in the latter case we use ideas from linear algebra.

The reader should note that we will say nothing about any supposed relationship between connectionist networks and actual biology (our use of the term 'connectionist' as opposed to 'neural' is intended to emphasize this). Although we accept that much can be learnt by looking at the properties of actual neurons and synapses *etc*, it is not relevant to the study presented herein, which will be based entirely in the realms of engineering, mathematics and computer science.

1.2 Feedforward Connectionist Networks

The basic principles of pattern classification and function approximation, and the use of connectionist networks therein, are well documented and we will therefore not provide an exhaustive review. We assume that the reader has some prior knowledge of these areas, including for example knowledge of statistical pattern classification, Bayes decision theory, standard connectionist networks such as the MLP, and an understanding of the concept of supervised learning. In this section we provide a brief introduction, mainly for the purpose of providing some basic notation and nomenclature for what follows. Further detail can be found in Duda and Hart [4], Lippmann [5, 6] and Hush and Horne [7].

1.2.1 Pattern Classification

In pattern classification, we are interested in assigning sets of measurements, taken in a given situation, to specific classes. The set of all possible measurements forms an *input space* $\mathcal{I}$; it is usual to deal with $\mathcal{I} = \mathbb{R}^n$ for some positive integer n and this is the case in the following work. A specific measurement provides us with an *input vector* $\mathbf{x} \in \mathcal{I} = \mathbb{R}^n$, the elements of which in general correspond either to measurements taken directly from the real world — pressure, temperature or other physical quantities, or perhaps the pixel values of an image — or to *features* derived therefrom. We wish to map input vectors $\mathbf{x} \in \mathcal{I}$ to *pattern classes*. For a problem involving q classes we typically have a set $\mathcal{C} = \{c_1, c_2, \ldots, c_q\}$ of q class indices and hence this process can be modelled by a mapping $f : \mathcal{I} \to \mathcal{C}$.

As a very simple example of this type of problem, for which $q = 2$, consider input vectors

$\mathbf{x} \in \mathcal{I}$ which contain the pixel values of an image of an apple. We require that the output of the system is c_1 if the apple is rotten and c_2 otherwise.

1.2.2 Connectionist Networks

Clearly the design of a pattern classifier of the type described above can be regarded as the design of a device which implements a mapping $f : \mathbb{R}^n \to \mathcal{C}$; the connectionist network techniques in which we are interested provide one of many alternative ways of designing and implementing the desired mapping.

The main distinguishing feature of a connectionist network when compared to the available alternatives is its architecture. There is no standard and generally accepted definition of a connectionist network, and we will not attempt to introduce one. It is sufficient for our purposes to define a connectionist network as a system of interconnected processors (the interconnections in general being directed). The network has n inputs which will generally correspond to the elements of an input vector $\mathbf{x} \in \mathbb{R}^n$, and one or more outputs which indicate the class corresponding to any input $\mathbf{x}$. The processors, or *nodes*, are relatively simple[1], and each processor p_i has an associated vector $\mathbf{w}_i$ of parameters or *weights* which can be used to alter the specific function $g_{\mathbf{w}_i}$ that it computes. We will be concerned with parameter vectors $\mathbf{w}_i$ having real-valued elements, although other types have also been used. Similarly, the functions $g_{\mathbf{w}_i}$ can have any appropriate domain and range, although we will in general deal with domains which are Euclidean spaces and ranges which are either Euclidean spaces or the set $\{+1, -1\}$. In the following work we are interested only in feedforward networks, meaning simply that our networks contain no cycles. A typical network is illustrated in figure 1.1; the network as a whole clearly computes a function $f_{\mathbf{w}} : \mathbb{R}^n \to \mathcal{C}$, where $\mathbf{w}$ is a vector containing all of the available variable weights.

1.2.3 Connectionist Networks as General Function Approximators

Connectionist networks are also commonly used in order to perform more general function approximation tasks. For our purposes these cases can be modelled by allowing the set of class indices to be $\mathcal{C} = \mathbb{R}$, such that the network computes some function $f_{\mathbf{w}} : \mathbb{R}^n \to \mathbb{R}$. This type of scenario arises, for example, if we wish to use the network in order to predict a future value of some appropriate time series, such as a sampled speech signal, given the most recent n sample values.

1.2.4 Supervised Learning

Obviously the actual function computed by a connectionist network can be altered if we alter any of the parameter vectors $\mathbf{w}_i$ associated with the individual processors; we denote by $\mathcal{F}$ the class of all functions $f_{\mathbf{w}}$ that can be computed by a network. The task of designing a network involves three choices: the choice of the class of functions computed by each node, the choice of the specific architecture used, and the choice of the actual values for the parameters. We are interested in the design of networks using *supervised learning.* This means that we have access to a *training sequence*,

$$T_k = ((\mathbf{x}_1, o_1), (\mathbf{x}_2, o_2), \ldots, (\mathbf{x}_k, o_k)) \tag{1.1}$$

of k *training examples* where $\mathbf{x}_i \in \mathbb{R}^n$ are inputs to the network and $o_i \in \mathcal{C}$ are the corresponding classifications. The sequence T_k will typically be constructed by collecting actual inputs $\mathbf{x}$ for

[1]'Simple' is of course a term which is difficult to define in this context. Examples of particular functions appear in later chapters.

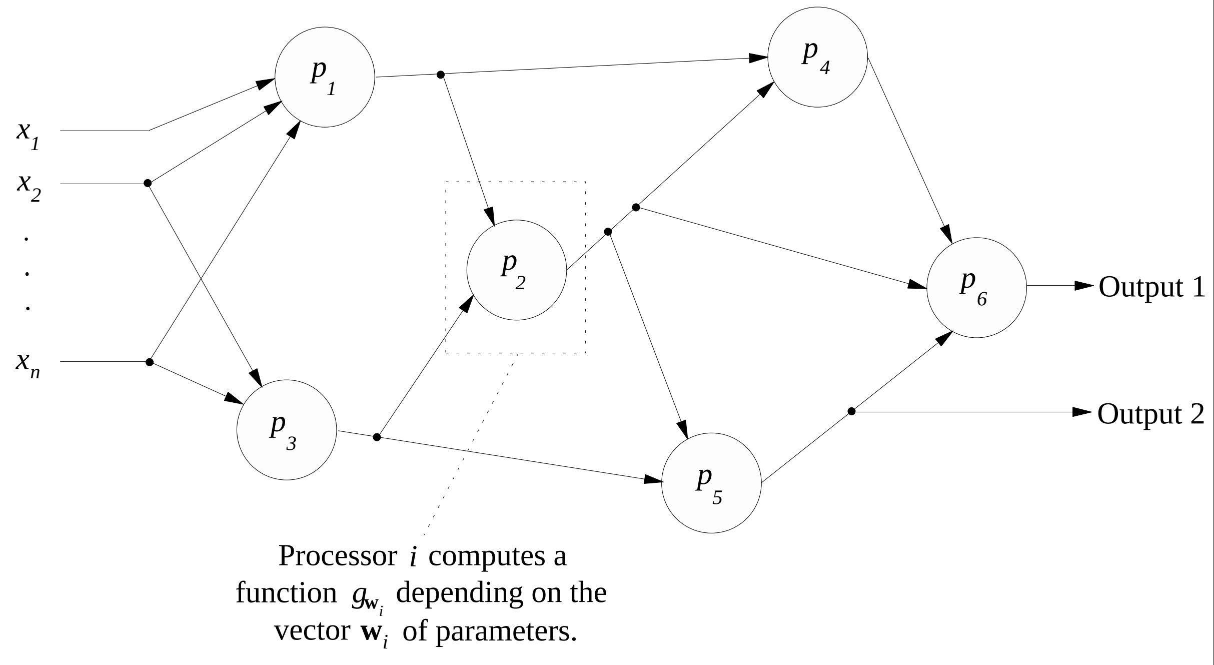

x_1
x_2
x_n
p_1
p_2
p_3
p_4
p_5
p_6
Output 1
Output 2
Processor i computes a function $g_{\mathbf{w}_i}$ depending on the vector $\mathbf{w}_i$ of parameters.

Figure 1.1: A typical feedforward connectionist network with two outputs. Inputs are elements of a vector $\mathbf{x} \in \mathbb{R}^n$.

which the correct classification is known, and in general includes only a subset of the possible inputs to the network. It may either provide a true representation of the underlying problem or, more realistically, be corrupted by noise. The network is designed entirely on the basis of the available examples and any *a priori* information about the problem that we wish to incorporate. We typically attempt to construct a network such that an appropriate measure of its *error* with respect to the training sequence is minimized; this idea is made more precise in the next chapter. Once the network has been trained, we hope that it exhibits some ability to *generalize*.

The ability of a network to generalize is important, and the theoretical investigation of the generalization ability of a particular class of networks forms the first of the two major aims of this dissertation. In order to motivate the study of generalization, we now provide a summary of the issues involved.

1.3 Generalization

In an historical context, the fact that connectionist networks usually exhibit some ability to perform generalization has always been perceived as one of their most important properties, and it is fair to say that this ability has been one of the major reasons for the considerable degree of interest shown in the subject. Although there is no universally accepted definition of the term 'generalization', it is invariably used, in the context of pattern classification, to describe the ability of a connectionist network or other pattern classifier to correctly classify an input vector which was not present in the set of examples used to train the system.

1.3.1 Experimental and Theoretical Studies of Generalization: Why do we Need Theoretical Studies?

Most research performed on generalization has been, until relatively recently and with the exception of a relatively small number of studies, of an experimental nature. Typically, a particular network has been trained on a particular set of data from a particular problem and then was said to 'generalize' in some sense if it performed well when tested using an independent set of test data from the same problem. If the network performs well in such an experiment, we have some evidence of generalization ability, but only in relation to the *specific* network architecture and node functions used, and the *specific* problem addressed. Of course, this approach is quite acceptable if we can find a network that appears to generalize 'well enough' for the particular problem we wish to solve, but then some obvious questions present themselves: will the network be guaranteed to provide good generalization for examples from the same problem that weren't in the set of test data? Or: does a different network exist which will generalize better? and so on. In short, a purely experimental approach is unlikely to be fruitful if we wish to answer more *general*, rather than *specific*, questions about generalization ability, and to gain a true understanding of the phenomenon.

One of the reasons that the quantity of theoretical work published is relatively small is perhaps the difficulty of *defining*, in an exact manner, what is meant by 'good generalization', and several definitions have now appeared. It is worth noting that, in one sense, the concept of 'good generalization' is rather arbitrary, and that it depends entirely on the precise nature of the problem being addressed *and on the views of the user*; in this case analysis can only be attempted for specific cases. A similar observation has been advanced by Minsky and Papert [3, pages 278–279].

It is also worth noting that, when considering generalization, there are some problems which *no* pattern classifier, connectionist network or otherwise, can solve without being trained using every possible input/output pair. An example given by Girosi and Poggio [8] is that of learning to associate names in a telephone directory with their telephone number. No matter how large a subset of the possible set of examples we use to train the network its chances of generalizing are very small. In general, any mapping which is completely local or random in this manner will cause problems.

1.3.2 Generalization and *A Priori* Knowledge

A generally perceived advantage of connectionist networks is that they do not usually need to be supplied with a great deal of *a priori* knowledge regarding the problem to be solved. It is very much an open problem to what extent it will in general be necessary to incorporate *a priori* knowledge of a particular problem when designing a network in order to obtain good generalization, and of course, any network architecture and training algorithm which do not require this knowledge would be highly desirable. Again, this problem clearly cannot be satisfactorily solved using experimental studies alone.

1.3.3 Generalization and Network Architecture

In addition to gaining a good understanding of the generalization ability of particular types of network, it would be useful to be able to make general comparisons between different types of connectionist network. For example, we would like to be able to ask questions such as: is a network of type A *inherently* better at performing generalization than a network of type B? Or: does a network of type A typically require fewer training examples to achieve a given generalization performance than a network of type B? Questions such as this clearly require theoretical, as well as experimental analysis if they are to be answered adequately. Furthermore, we would like to know precisely what effect the architecture of a network has on its ability to generalize, as perhaps the most important result to appear consistently to date in experimental studies has been that the 'size' of a network must be chosen correctly if we are to obtain good generalization; once again, this question needs theoretical as well as experimental analysis.

In fact, there are further reasons for which the selection of an appropriate architecture is important and it is an investigation of this problem for the particular class of networks of interest which forms the second major aim of this dissertation.

1.4 The Selection of a Suitable Network Architecture

In reviewing the existing literature on connectionist network training algorithms, it quickly becomes clear that the majority of the research published to date is dedicated to the study of networks having a *fixed architecture*. That is, the number of processors in the network and the pattern of interconnections between them is forced to remain fixed throughout the process of training while only the values of the weights are adapted in response to the presentation of training examples.

This is the case in particular in the early published literature, with a small number of exceptions such as the Group Method of Data Handling (GMDH) introduced by Ivakhnenko [9] and the statistical techniques reviewed by Barron and Barron [10]. In the more recent literature the fact that the actual architecture used is important, and should be carefully chosen, has found considerably more acceptance, and more research has begun to appear in what we will call *self-structuring* training algorithms. These algorithms choose an architecture which is in some sense optimum, as well as the appropriate corresponding weights, during training. As we might expect, research to date has been aimed mostly at MLP type networks.

Why is self-structuring important? Three major reasons can be identified, relating to computational load, storage, and generalization ability. In all three cases we benefit in general from using the *smallest* network capable of learning to perform satisfactorily on a set of training examples, where the size of the network is measured in an appropriate manner.

Firstly, consider computational load. After training, the time taken for a feedforward connectionist network to classify a new input will generally be affected by its size. This is particularly true of multilayer networks in which the time taken increases with the number of layers. We also need to consider the time required to determine the values of the weights, which generally increases with increasing network size. Secondly, the amount of storage required in implementing a network clearly depends directly on the number of weights and nodes, and on the complexity of the nodes. Finally, we will see that, in general, in order to obtain a network with good generalization performance for a particular problem it is a good strategy to use the smallest network capable of learning the training examples to a sufficient accuracy. This can be regarded as a simple application of the principle of *Occam's Razor*; in standard statistical parlance it can be regarded as an attempt to avoid *overfitting* the available data.

There is also one further good reason for using self-structuring. This is that the imposition of a fixed architecture before training proceeds can be regarded as the imposition of *a priori* knowledge about the problem being addressed; it can be thought of as being similar to *inductive bias* in artificial intelligence. If *a priori* knowledge which allows the user to make a sensible choice of architecture in this manner is available then obviously it is sensible to use it. However, it is usually the case in the literature that an *arbitrary* choice of fixed architecture is made. In any case, as we have already indicated, one of the major hopes for connectionist networks has always been that eventually it may be possible to apply them to reasonably general problems without having to supply this kind of *a priori* knowledge, and the study of self-structuring provides an obvious method with which we can try to achieve this aim.

1.5 Organization of the Dissertation

This dissertation is divided into three parts. Part I consists of a single chapter (chapter 2) in which we introduce and review the class of networks with which the remainder of our work is concerned: the class of Φ-*networks*.

In part II we address the ability of Φ-networks to generalize by performing a comprehensive theoretical analysis. The vehicle for our analysis is an area of computational learning theory based on the theory of *Probably Approximately Correct (PAC) learning*, and chapter 3 provides a full introduction to the required techniques along with a brief discussion of the alternative methods that are available for analysing generalization. In order to apply the theory introduced in chapter 3 we need to calculate bounds on two quantities associated with a connectionist network, called the *growth function* and the *Vapnik-Chervonenkis (VC) dimension*, for the case of interest, namely the class of Φ-networks. This problem is addressed in chapter 4, in which we calculate such bounds for the general class of Φ-networks as well as for various special cases thereof. This chapter also uses the idea that the VC dimension can be regarded as a measure of capacity to provide an explanation for the fact that Φ-networks can in practice require a relatively large number of weights compared to multilayer perceptrons. In chapter 5 we use the results of the previous two chapters to derive necessary and sufficient conditions on the number of training examples required in training a Φ-network such that a particular generalization performance can be expected. We then compare our results to comparable results derived elsewhere for multilayer perceptron type networks, and we attempt to model the effect of the use of self-structuring training algorithms on the sufficient conditions.

In part III, which consists of a single chapter (chapter 6), we address the problem of constructing a good self-structuring training algorithm for Φ-networks. Having discussed the best way in which to define the 'optimum' architecture for a Φ-network, we use the fact that training such a network when the architecture is fixed can be regarded as a *least squares* problem to derive a training algorithm that searches for a network with a good architecture. Our analysis of the problem allows us to show that the well-known *weight decay* method for performing self-structuring does not necessarily provide an optimum architecture. We then test our algorithm experimentally, before extending our analysis of the problem in a way that we expect will allow an improved version of our algorithm to be developed.

Finally, chapter 7 of the dissertation provides conclusions and suggestions for future research.

Part I

Φ-Networks

Chapter 2

Φ-Networks

2.1 Introduction

This chapter introduces the class of Φ-networks, henceforth called Φ-nets, which is similar to a type of network introduced originally in the early 1960s. These networks have also been called *Linearly Weighted Connectionist Networks*, for reasons which will become clear, and it is the properties of this class of networks with which this dissertation is primarily concerned.

The most popular and most frequently used feedforward connectionist network at present is the multilayer perceptron trained using hidden layer back-propagation (HLBP), or some variant thereon. This type of network has several well-known shortcomings which we summarize in section 2.3. One of the original motivations for investigating Φ-nets was the observation that particular members of this class appeared in practical studies to offer major advantages over MLPs, especially in terms of the time required for training, but that, surprisingly, they tended to be used by only a minority of researchers. This chapter, which consists partly of review material and partly of original comment, is in part an attempt to argue that Φ-nets with *fixed basis functions* have significant advantages when compared to MLPs; the promotion of the class of Φ-nets can be regarded as a subsidiary aim of the chapter.

The class of Φ-nets is introduced as a *unifying formalism* which allows us to investigate several individual past and contemporary network types contained within the overall class — each of which has previously been extensively studied in its own right — without needing to resort to a separate analysis for each type. Section 2.2 introduces the Φ-net formalism, section 2.3 introduces the required material on MLPs, and section 2.4 reviews training algorithms for Φ-nets. In section 2.5 we show that many standard connectionist networks are special cases of the class of Φ-nets — a fact that has not been widely appreciated previously, especially in the case of regularization networks, modified Kanerva models and cerebellar model articulation controllers (CMACs), although it has recently also been noted by Boser *et al.* [11] and Renals and Rohwer [12] for some types of network. As it is these individual networks, rather than the class of Φ-nets *per se*, which provided the motivation for this work, we also review the existing literature on the networks which are the most extensively studied examples of Φ-nets to date. In section 2.6 we discuss the relation of Φ-nets to other types of network used to solve similar problems, and we further motivate the study of Φ-nets with fixed basis functions by providing some examples of practical problems to which they have been successfully applied. We also introduce and discuss some possible criticisms of Φ-nets. Section 2.7 concludes the chapter.

In common with most research areas in the field of connectionist networks, this area has a vast associated past and contemporary literature, and an encyclopædic review would not be feasible or appropriate. This chapter therefore summarizes what we consider to be the most important and pertinent work in the context of this dissertation, and attempts to give a representative sample of the published theoretical and experimental results for the different types of Φ-net. An excellent review of the various alternative types of connectionist network is given by Lippmann [5, 6], see also Hush and Horne [7], Bressloff and Weir [13], Pao [14] and Hertz *et al.* [15].

2.2 The Class of Φ-Nets

2.2.1 The Definition of a Φ-Net

A Φ-net takes as input a vector $\mathbf{x} \in \mathbb{R}^n$ where,

$$\mathbf{x}^T = [\ x_1 \quad x_2 \quad \cdots \quad x_n\ ] \tag{2.1}$$

and produces as output a class index $f(\mathbf{x}) \in \mathcal{C}$ where $\mathcal{C}$ is a set of class indices. In the following chapters two types of Φ-net are studied, differing only in the set $\mathcal{C}$ used. In the first type, which would be used for example in an application such as time series prediction (Lowe and Webb [16]), we use $\mathcal{C} = \mathbb{R}$, and in the second — intended specifically for the study of two-class pattern classification problems — we use $\mathcal{C} = \mathbb{B} = \{+1, -1\}$. The networks being considered have only a single output; we discuss possible extensions of our work to networks with many outputs at appropriate points in the text. A Φ-net can thus be thought of as implementing some, typically nonlinear, mapping $f : \mathbb{R}^n \rightarrow \mathcal{C}$.

A general Φ-net, for which the structure is shown in figure 2.1, is defined by a 5-tuple $\mathcal{N} = (n, m, \Phi, \mathbf{w}, \mathcal{C})$ in which,

- $n \geq 1$ is the number of inputs to the network.
- Φ is a set of $m =\mid \Phi \mid \geq 1$, typically nonlinear, real-valued *basis functions* with domain $\mathbb{R}^n$. Formally,
$$\Phi = \{\phi_i(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R} \mid i = 1, 2, \ldots, m\}. \tag{2.2}$$
We always assume that the basis functions are *fixed* unless otherwise stated. Although we will mostly consider only fixed basis functions, we do briefly consider at some points Φ-nets in which the basis functions can be adapted.
- $\mathbf{w} \in \mathbb{R}^{m+1}$ is a vector of weights,
$$\mathbf{w}^T = [\ w_0 \quad w_1 \quad w_2 \quad \cdots \quad w_m\ ]. \tag{2.3}$$
We define $W = m + 1$ to be the number of variable weights used by the network.
- $\mathcal{C}$ is the set of class indices introduced above.

The set Φ of basis functions is used in order to construct a, generally fixed, mapping $\varphi : \mathbb{R}^n \rightarrow \mathbb{R}^{m+1}$, $\varphi : \mathbf{x} \mapsto \tilde{\mathbf{x}}$ which maps input vectors $\mathbf{x}$ into *extended vectors*[1] $\tilde{\mathbf{x}}$. The extended vector is formed as[2],

$$\tilde{\mathbf{x}}^T = [\ 1 \quad \phi_1(\mathbf{x}) \quad \phi_2(\mathbf{x}) \quad \cdots \quad \phi_m(\mathbf{x})\ ], \tag{2.4}$$

[1]The term 'extended' is used as often $m + 1 > n$. However, this is *not* a definite requirement and if necessary $m + 1 \leq n$ can be used.

[2]Not all of the specific examples of Φ-nets introduced later in this chapter use a unit term in their extended vector as standard. We assume that this term is always included as it is important for some of our later analysis and as its inclusion is useful in general.

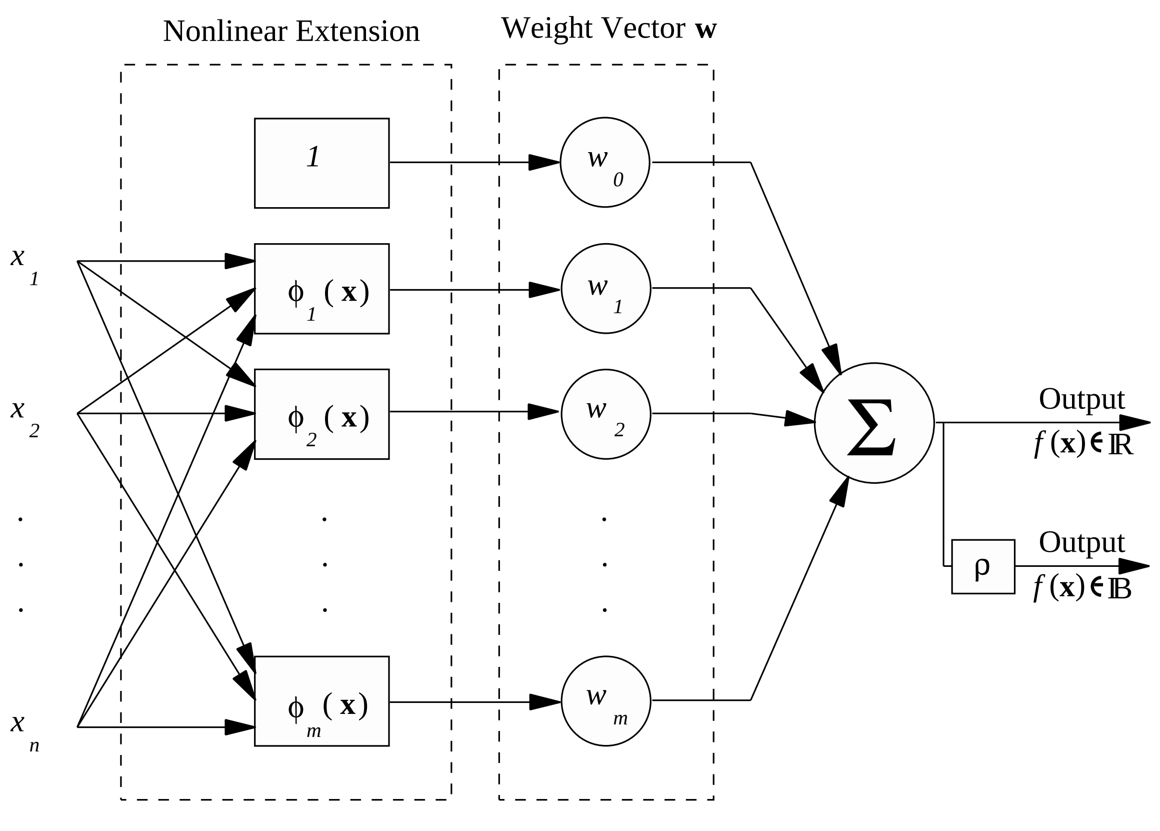


Figure 2.1: The structure of a general Φ-net.

and we call the space $\mathbb{R}^m$ containing vectors $[\ \phi_1(\mathbf{x})\ \ \phi_2(\mathbf{x})\ \ \cdots\ \ \phi_m(\mathbf{x})\ ]^T$ the *extended space*. For the case where $\mathcal{C} = \mathbb{R}$ the output of the network is formed as,

$$\begin{aligned} f_{\mathbf{w}}(\mathbf{x}) &= \mathbf{w}^T\tilde{\mathbf{x}} \\ &= \mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}) \\ &= w_0 + \sum_{i=1}^{m} w_i\phi_i(\mathbf{x}) \end{aligned} \tag{2.5}$$

where we call the mapping implemented by the network $f_{\mathbf{w}}$ in order to emphasize that it depends on the parameters specified by the weight vector $\mathbf{w}$. Examining equation 2.5, and recalling that the functions ϕ_i are fixed, the origin of the expression 'linearly weighted connectionist network' should now be clear. For the case of $\mathcal{C} = \mathbb{B}$ the output of the network is,

$$f_{\mathbf{w}}(\mathbf{x}) = \rho(\mathbf{w}^T\tilde{\mathbf{x}}) \tag{2.6}$$

where[3],

$$\rho(x) = \begin{cases} +1 & x \geq 0 \\ -1 & x < 0 \end{cases} . \tag{2.7}$$

In the latter case we call the network a *threshold Φ-net*.

The fact that, in equation 2.5, the output $f_{\mathbf{w}}$ is clearly a *linear combination* of the basis functions and the unit term, is probably the single most important property of the class of Φ-nets, and is discussed below. Although we will be concerned almost entirely with Φ-nets having a single output, the advantages resulting from this linearity also carry over to the multiple output case.

For the case where $\mathcal{C} = \mathbb{B}$ an important interpretation is available of the way in which this type of network operates. Nonlinear decision boundaries are constructed in the input space by mapping input vectors into a new space, typically of higher dimension, *such that they are linearly separable*. The network divides the new space — the extended space — into two halfspaces, the decision boundary in this space being the hyperplane,

$$\mathbf{w}^T\tilde{\mathbf{x}} = 0. \tag{2.8}$$

This is illustrated in figure 2.2.

This definition of a Φ-net is similar to definitions for networks which appeared in the 1960s, see for example Duda and Hart [4], Cover [17] and Nilsson [18]. It is also closely related to the definition of the more recently introduced *functional-link net* [14].

2.2.2 Function Classes and Graph Classes

Clearly, given a specific weight vector $\mathbf{w}$ a Φ-net implements a particular mapping $f_{\mathbf{w}} : \mathbb{R}^n \rightarrow \mathcal{C}$; the actual mapping implemented can be altered by altering $\mathbf{w}$. We define $\mathcal{F}_n^{\Phi}$ as the class of all mappings that a particular Φ-net can implement. Formally,

$$\mathcal{F}_n^{\Phi} = \{f_{\mathbf{w}} \mid \mathbf{w} \in \mathbb{R}^{m+1}\}. \tag{2.9}$$

The class $\mathcal{F}_n^{\Phi}$ will be used only where $\mathcal{C} = \mathbb{B}$ unless otherwise stated. Similarly, any mapping $f_{\mathbf{w}}$ can be associated with a graph $\Gamma_{\mathbf{w}} \subset \mathbb{R}^n \times \mathcal{C}$ where,

$$\Gamma_{\mathbf{w}} = \{(\mathbf{x}, f_{\mathbf{w}}(\mathbf{x})) \mid \mathbf{x} \in \mathbb{R}^n\} \tag{2.10}$$

[3]Some definitions of $\rho(x)$ for similar types of network require that ρ is *undefined* for $x = 0$. Our definition is more convenient, being properly defined for all values of x, and in general the results obtained are the same in both cases.

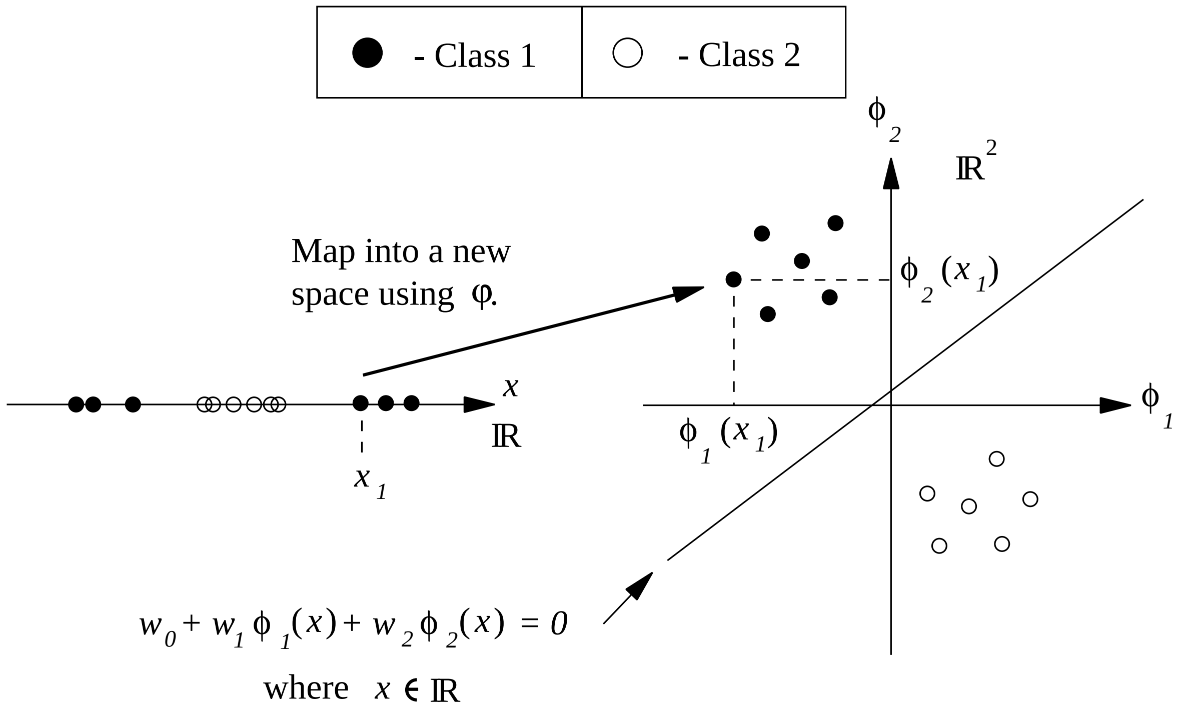


Figure 2.2: Input vectors are mapped into a new space such that they are linearly separable.

and we define the class of graphs Γ_n^Φ as,

$$\Gamma_n^\Phi = \{\Gamma_{\mathbf{w}} \mid \mathbf{w} \in \mathbb{R}^{m+1}\}. \tag{2.11}$$

2.2.3 Φ-Nets with Adapting Basis Functions

Although we mostly consider Φ-nets with fixed basis functions, for reasons which will be discussed below, it will at some points be necessary to discuss Φ-nets with adapting basis functions. In this case, each basis function $\phi_i(\mathbf{x}; \mathbf{y}_i)$ depends on a vector,

$$\mathbf{y}_i^T = \left[\begin{array}{cccc} y_1^{(i)} & y_2^{(i)} & \cdots & y_{p_i}^{(i)} \end{array} \right] \tag{2.12}$$

of p_i, typically real-valued, parameters, and the network implements a mapping $f_{\mathbf{w},\boldsymbol{\theta}} : \mathbb{R}^n \rightarrow \mathcal{C}$ which depends on both $\mathbf{w}$ and the parameter vector,

$$\boldsymbol{\theta}^T = \left[\begin{array}{cccc} \mathbf{y}_1^T & \mathbf{y}_2^T & \cdots & \mathbf{y}_m^T \end{array} \right]. \tag{2.13}$$

The class of functions $\mathcal{F}_n^\Phi$ for this type of Φ-net is defined as,

$$\mathcal{F}_n^\Phi = \left\{ f_{\mathbf{w},\boldsymbol{\theta}} \mid \mathbf{w} \in \mathbb{R}^{m+1}, \boldsymbol{\theta} \in \mathbb{R}^p \text{ where } p = \sum_{i=1}^m p_i \right\}. \tag{2.14}$$

2.3 The Multilayer Perceptron and the HLBP Algorithm

The multilayer perceptron has a feedforward structure, as illustrated in figure 1.1, in which node i computes a function,

$$g_{\mathbf{w}_i}(\mathbf{z}_i) = \sigma \left[w_0^{(i)} + \sum_{j=1}^{n_i} w_j^{(i)} z_j^{(i)} \right] \tag{2.15}$$

where $\mathbf{z}_i^T = [\ z_1^{(i)} \ \ z_2^{(i)} \ \ \cdots \ \ z_{n_i}^{(i)}\ ]$ is a vector of n_i inputs to the node,

$$\mathbf{w}_i^T = [\ w_0^{(i)} \ \ w_1^{(i)} \ \ \cdots \ \ w_{n_i}^{(i)}\ ] \tag{2.16}$$

is a vector of variable weights associated with the node, and σ is an *activation function.* The function σ is typically either a *sigmoid* of a form such as,

$$\sigma(x) = s_1(x) = \frac{1}{1 + \exp(-\beta x)} \text{ for } x \in \mathbb{R} \text{ and } \beta \in \mathbb{R}^+ \tag{2.17}$$

or

$$\sigma(x) = s_2(x) = \tanh(\beta x) \text{ for } x \in \mathbb{R} \text{ and } \beta \in \mathbb{R}^+ \tag{2.18}$$

where β is a 'steepness' parameter, or a step function $\sigma = \rho$. The network typically has a *layered* structure as illustrated in figure 2.3. There is at present no generally accepted convention regarding what is regarded as a 'layer' in such a network; for example, the inputs may or may not be regarded as a layer. We will take the latter point of view, and hence refer to the network in figure 2.3 as a *three-layer* or *two hidden-layer* network. More general connections between inputs and processors than are implied by this figure are also often permitted; for example, connections between inputs and nodes in hidden layer 2, or connections between nodes in hidden layer 1 and the output layer. We call any node which is in a hidden layer a *hidden node.*

When the activation function σ is of the form of equation 2.17 or 2.18 the network is usually trained using the hidden layer back-propagation (HLBP) algorithm or some variant thereon. The

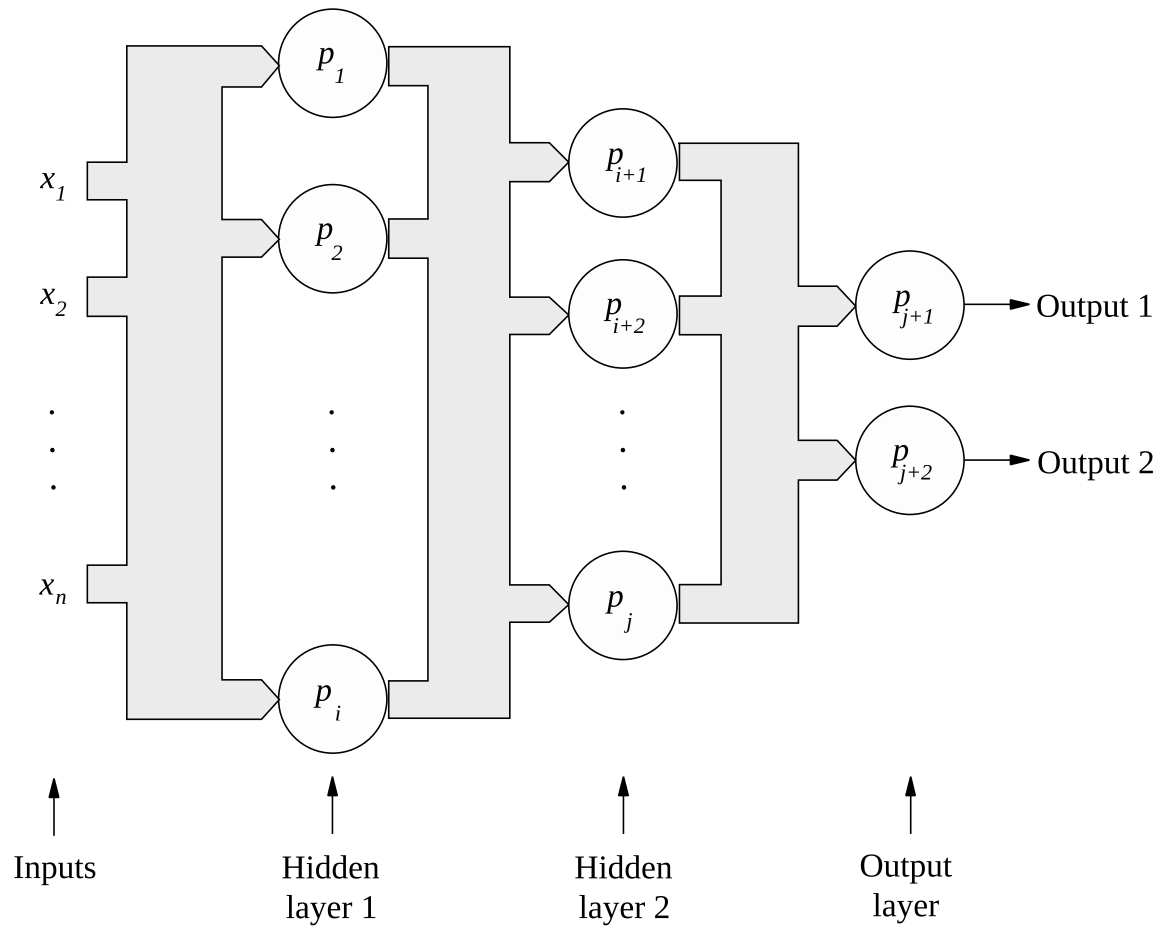


Figure 2.3: The layered structure of a standard multilayer perceptron. We refer to this network as a *three-layer* or *two hidden-layer* network.

standard HLBP algorithm is well documented (see for example Rumelhart and McClelland [19]) and so we do not include the details. What is important for our purposes is to note that MLPs have well-known and significant limitations. In particular we note, firstly, that the error surface of an MLP is multimodal, and consequently it is in general difficult to devise training algorithms which can successfully avoid convergence to a local minimum. Secondly, we note that, regardless of the training algorithm used, the training process for an MLP tends to be relatively time-consuming, and the time taken scales badly as the size of the network is increased. This observation is reinforced by several theoretical results which prove that training an MLP is hard in the computational sense (Garey and Johnson [20]). Specifically, the problem is **NP**-complete, which implies that it is computationally intractable under the usual assumption that $\mathbf{P} \neq \mathbf{NP}$. Results of this nature can be found in Judd [21, 22], Blum and Rivest [23], Kolen and Goel [24] and Orponen [25].

In this dissertation we will mostly deal with MLPs having step-type activation functions $\sigma = \rho$, as this allows us to make comparisons with Φ-nets. Note that the computational complexity results still apply when this type of activation function is used. We discuss at appropriate points the effect on our results of using more usual activation functions such as s_1 and s_2.

2.4 Training Algorithms for Φ-Nets with Fixed Basis Functions

There are many highly developed training algorithms for Φ-nets with fixed basis functions, and in this section we provide a brief review. For the time being, we consider training only in terms of the minimization of the error of a network with respect to a particular sequence of training examples. It is of course possible to introduce more sophisticated criteria for training, and this is part of the aim of chapter 6. In most training algorithms, training consists of choosing an optimum weight vector $\mathbf{w}_{\text{opt}}$ which minimizes an appropriately defined measure of error which we denote by $\xi(\mathbf{w})$.

2.4.1 Networks with $\mathcal{C} = \mathbb{R}$

Consider a Φ-net with $\mathcal{C} = \mathbb{R}$, and assume we have a sequence,

$$T_k = ((\mathbf{x}_1, o_1), (\mathbf{x}_2, o_2), \ldots, (\mathbf{x}_k, o_k)) \tag{2.19}$$

of k training examples where $\mathbf{x}_i \in \mathbb{R}^n$ and $o_i \in \mathbb{R}$ for $i = 1, 2, \ldots, k$. In general, we will have $k > W = m + 1$. If the training sequence is to be learnt exactly we would like to choose a weight vector $\mathbf{w}$ such that $f_{\mathbf{w}}(\mathbf{x}_i) = o_i$ for $i = 1, 2, \ldots, k$. Equivalently, we would like to solve the set of equations,

$$\mathbf{P}\mathbf{w} = \mathbf{o} \tag{2.20}$$

where,

$$\mathbf{P} = \begin{bmatrix} \boldsymbol{\varphi}^T(\mathbf{x}_1) \\ \boldsymbol{\varphi}^T(\mathbf{x}_2) \\ \vdots \\ \boldsymbol{\varphi}^T(\mathbf{x}_k) \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{x}}_1^T \\ \tilde{\mathbf{x}}_2^T \\ \vdots \\ \tilde{\mathbf{x}}_k^T \end{bmatrix} \tag{2.21}$$

and $\mathbf{o}^T = [\ o_1 \quad o_2 \quad \cdots \quad o_k\ ]$. If the training sequence cannot be learnt exactly (which is in fact desirable in some cases) then we usually choose $\mathbf{w}$ such that the error $\xi(\mathbf{w}) = \|\mathbf{P}\mathbf{w} - \mathbf{o}\|^2$ is minimized, where $\|.\|$ is the Euclidean norm. In other words, the problem of training a Φ-net can be re-cast as a linear least squares problem. This type of problem is well understood, and various algorithms are available for its solution; a comprehensive summary is given by Press *et al.* [26]. We return to this approach to training, and discuss it in further detail, in chapter 6.

In a similar manner, we can attempt to minimize the expected value of the squared error,

$$\xi(\mathbf{w}) = \mathrm{E}\left[(o_i - \mathbf{w}^T\tilde{\mathbf{x}}_i)^2\right] \tag{2.22}$$

and this leads to the well-known *Least Mean Squares* (LMS) algorithm (see Haykin [27]). Again, we discuss this approach further in chapter 6.

2.4.2 Networks with $\mathcal{C} = \mathbb{B}$

Consider now a network with $\mathcal{C} = \mathbb{B}$, and assume again that we have a training sequence T_k, where in this case $\mathbf{x}_i \in \mathbb{R}^n$ and $o_i \in \mathbb{B}$ for $i = 1, 2, \ldots, k$. Again, we may wish to select a weight vector $\mathbf{w}$ such that $f_{\mathbf{w}}(\mathbf{x}_i) = o_i$ for each training example. If it is possible to do this we say that the training examples are *separable*; this occurs if the examples are linearly separable in the extended space.

There are many algorithms available for training this type of network, the most common of which are discussed in [4]. The algorithms typically have associated convergence results; for example, in some cases an algorithm will always obtain, in finite time, a weight vector which correctly classifies all the training examples provided they are linearly separable in the extended space. For sequences of training examples which are *not* separable, algorithms such as those introduced above for $\mathcal{C} = \mathbb{R}$, linear programming algorithms, or the *Ho-Kashyap Procedure* [4] can be employed to obtain approximate solutions.

2.4.3 Further Comments on Training Algorithms

The most important difference between the training algorithms mentioned in this section and training algorithms for many other networks, including MLPs, is that the linearity inherent in Φ-nets with fixed basis functions allows their training algorithms to run relatively quickly in most cases. Significant reductions in training time have been reported in practical comparisons with MLPs trained using HLBP (Lynch and Rayner [28]). Of course, difficult situations exist which can make training difficult, but our observation appears to be widely valid on the basis of the available experimental results. It is also important to note that when using the training algorithms mentioned for $\mathcal{C} = \mathbb{R}$, we are usually able to deal with situations in which, as we shall see in chapter 6, either the error surface is unimodal, or for which the structure of the solution space is well-defined and highly tractable. We therefore do not suffer from problems caused by local minima.

In the next section, we will consider some specific Φ-nets in which the basis functions have parameters associated with them which we in general regard as being *fixed*. It is obviously tempting to adapt these parameters as well as $\mathbf{w}$ during training, and indeed some researchers have done this. However, this will in general turn the problem into one of nonlinear optimization, and we consequently lose the advantages mentioned above. Lowe [29] has made some particularly pertinent observations regarding this approach, which we discuss in subsection 2.5.3.

Training algorithms have also been suggested in which a simple criterion is used to select parameters associated with the basis functions. These parameters are then *fixed* allowing the training algorithms presented above to be applied *separately*. This distinction is important, and we will discuss it at length in chapter 5. Algorithms of this type have been applied with considerable success (see for example Moody and Darken [30]) suggesting that to some extent they allow us to obtain the advantages of both linear and nonlinear optimization strategies. We suspect that in fact they provide us with a fast way of finding a global, or a 'good' local minimum on the full error surface for the network which would otherwise need to be searched using nonlinear optimization.

Type of connectionist network	Form of basis functions
Linear discriminant function	$\phi_i(\mathbf{x}) = x_i$ where $i = 1, 2, \ldots, n$.
Perceptron	$\phi_i(\mathbf{x}) : \{0,1\}^n \rightarrow \{0,1\}$. The functions ϕ_i are *partial predicates*.
Polynomial discriminant function	$\phi_i(\mathbf{x})$ is a product of the elements of the input vector. For example $\phi_i(\mathbf{x}) = x_1 x_2^4 x_5^2$.
Radial basis function network	$\phi_i(\mathbf{x}) = \phi(\lVert\mathbf{x} - \mathbf{y}_i\rVert)$ where ϕ is a suitable function and $\mathbf{y}_i \in \mathbb{R}^n$ is a *centre*. Polynomial terms are also included in some cases.
Regularization network	$\phi_i(\mathbf{x}) = G(\mathbf{x}; \mathbf{y}_i)$ where G is a Green's function (see subsection 2.5.4) and $\mathbf{y}_i \in \mathbb{R}^n$ is a centre. Polynomial terms are also included in some cases.
Modified Kanerva model	$\phi_i(\mathbf{x}) = \begin{cases} 1 \text{ if } \mu_p(\mathbf{x}, \mathbf{y}_i) < r \\ 0 \text{ otherwise} \end{cases}$ where $\mathbf{y}_i \in \mathbb{R}^n$ is a *location*, r and p are fixed parameters and $\mu_p(\mathbf{x}, \mathbf{y}) = (\sum_{i=1}^n \mid x_i - y_i \mid^p)^{\frac{1}{p}}$.

Table 2.1: Summary of Basis Functions Used for Different Types of Φ-Net

2.5 Examples of Specific Networks in the Class of Φ-Nets

We now briefly review several connectionist networks and demonstrate that they are members of the class of Φ-nets. Linear and Polynomial discriminant functions, and perceptrons, have been studied extensively for several decades; radial basis function networks, regularization networks, and the modified Kanerva model were introduced more recently. The various networks introduced here differ only in the form of the basis functions used; table 2.1 provides a summary.

Most of the following discussion assumes that we use the set $\mathcal{C} = \mathbb{R}$ of class indices. This type of network can be converted to a two-class classifier with $\mathcal{C} = \mathbb{B}$ simply by adding to its output a unit implementing the function ρ of equation 2.7.

2.5.1 Linear Discriminant Functions and Perceptrons

Linear Discriminant Functions (LDFs) are probably the simplest type of pattern classifier available and are clearly members of the class of Φ-nets. In this case there are $m = n$ basis functions of the form,

$$\phi_i(\mathbf{x}) = x_i \text{ where } i = 1, 2, \ldots, m. \tag{2.23}$$

These classifiers have a large associated literature and many well-known limitations, see for example Minsky and Papert [3], Duda and Hart [4] and references therein, and Nilsson [18].

The class of *perceptrons*, originating in the work of Rosenblatt [31] and studied extensively in [3], is a subclass of the class of Φ-nets. In this case inputs are in $\{0,1\}^n$ rather than $\mathbb{R}^n$ and basis functions are *partial predicates* $\phi_i(\mathbf{x}) : \{0,1\}^n \rightarrow \{0,1\}$. It is important to note that many of Minsky and Papert's celebrated negative results rely on the use of partial predicates which are limited in ways which the Φ-net definition does not require; this is discussed further below.

2.5.2 Polynomial Discriminant Functions

Polynomial discriminant functions[4] (PDFs) are a natural extension of the LDFs. Of the Φ-nets which are capable of constructing nonlinear discriminant functions, these are probably the best known and most extensively studied. In this type of network functions $f_{\mathbf{w}}$ are of the form,

$$\begin{aligned}
f_{\mathbf{w}}(\mathbf{x}) &= w_0 + \sum_{i_1=1}^{n} w_{i_1} x_{i_1} \\
&\quad + \sum_{i_1=1}^{n} \sum_{i_2=i_1}^{n} w_{i_1 i_2} x_{i_1} x_{i_2} \\
&\quad + \cdots \\
&\quad + \sum_{i_1=1}^{n} \sum_{i_2=i_1}^{n} \cdots \sum_{i_d=i_{d-1}}^{n} w_{i_1 i_2 \cdots i_d} x_{i_1} x_{i_2} \cdots x_{i_d} \\
&= w_0 + \sum_{i=1}^{m} w_i \phi_i(\mathbf{x}), \qquad (2.24)
\end{aligned}$$

where the basis functions ϕ_i are in this case products of the elements of the input vector $\mathbf{x}$, for example, $\phi_i(\mathbf{x}) = x_1^3 x_2 x_{n-1}^5$. We call this network an (n, d) discriminator. It is possible to show that an (n, d) discriminator has $\Psi(n, d)$ weights where,

$$\Psi(n, d) = \left(\begin{array}{c} n+d \\ n \end{array} \right) = \frac{(n+d)!}{n!d!}, \qquad (2.25)$$

see Cover [17] or Casdagli [32]. In equation 2.25 we use the standard notation for the binomial coefficient,

$$\left(\begin{array}{c} i \\ j \end{array} \right) = \frac{i!}{j!(i-j)!}, \qquad (2.26)$$

which we will also sometimes write as C_j^i. The form of the summations in equation 2.24 prevents the same product of inputs from appearing many times, for example, $x_1 x_2$ and $x_2 x_1$ are assigned a single weight, rather than two separate ones.

Polynomial functions of the form of equation 2.24 have the desirable property that if Z is a compact subset of $\mathbb{R}^n$ and $C^0(Z)$ is the space of all continuous functions $f : Z \to \mathbb{R}$ then the set of all polynomial functions $p : Z \to \mathbb{R}$ is dense in $C^0(Z)$ (Chen *et al.* [33]). This type of network does however have some important disadvantages:

1. Polynomials can oscillate rapidly between training examples when used to model nonpolynomial nonlinearities.

2. Polynomial approximations are unstable under iteration which makes them unsuitable for tasks such as long term time series prediction (Lapedes and Farber [34]).

3. Examination of the form of the function Ψ shows that the number of weights required grows very quickly as n and d are increased. This is illustrated in figure 2.4.

A discussion of points (1) and (2) can be found in [32, 34]. The criticism of point (3) is well-known, however practical experience shows that good decision boundaries can in some cases be obtained using a reasonable number of coefficients (Specht [35]) or using small d (Rayner and Lynch [36]).

[4]For the case where $\mathcal{C} = \mathbb{B}$ we will call these functions *polynomial threshold functions*.

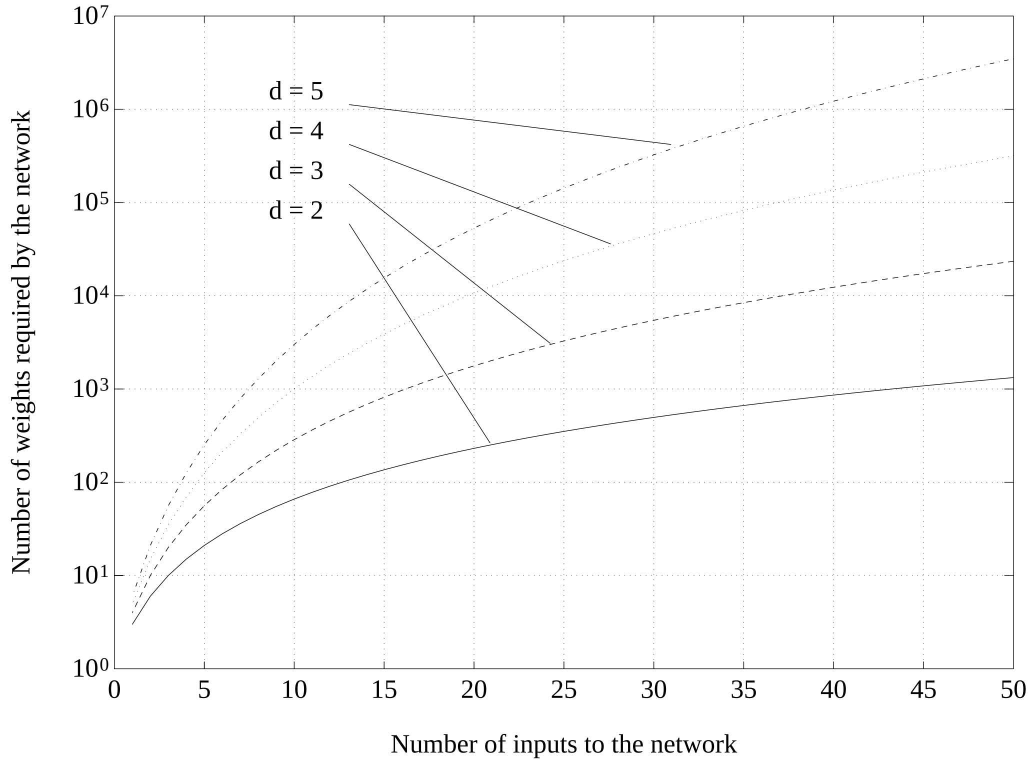


Figure 2.4: Behaviour of the function $\Psi(n, d)$ for different values of n and d.

Form of basis function	Type of basis function
$\phi_{LIN}(r) = r$	Linear
$\phi_{CUB}(r) = r^3$	Cubic
$\phi_{TPS}(r) = r^2 \ln r$	Thin plate spline
$\phi_{MQ}(r) = (r^2 + c^2)^{\frac{1}{2}}$, $c \in \mathbb{R}^+$	Multiquadric
$\phi_{IMQ}(r) = (r^2 + c^2)^{-\frac{1}{2}}$, $c \in \mathbb{R}^+$	Inverse multiquadric
$\phi_{GAUSS}(r) = \exp\left(-\left(\frac{r}{\sigma}\right)^2\right)$, $\sigma \in \mathbb{R}^+$	Gaussian

Table 2.2: Typical Basis Functions used in Radial Basis Function Networks

These networks have been studied by many authors since the 1960s, see for example Duda and Hart [4], Cover [17], Nilsson [18] and Maxwell *et al.* [37, 38], and the use of polynomials in closely related network types has a similarly rich history (Ivakhnenko [9] and Barron and Barron [10]). In [35] it is shown that PDFs arise naturally from an analysis of the pattern classification problem using nonparametric (Parzen window [4]) estimation of class-conditional probability density functions for use in the Bayes decision rule [4].

2.5.3 Radial Basis Function Networks

Radial basis function networks (RBFNs) were introduced by Broomhead and Lowe [39] on the basis of their observation that the theory of pattern processing using connectionist networks can be compared to the theory of multivariable interpolation in high dimensional spaces. Of the connectionist networks included in the class of Φ-nets, RBFNs are to date probably the most widely accepted.

In this approach the learning process is modelled as the task of making the network implement a mapping $f_{\mathbf{w}}$ (and corresponding graph $\Gamma_{\mathbf{w}}$) which is the 'best' approximation to a particular fixed mapping $f : \mathbb{R}^n \to \mathbb{R}$ with graph Γ. Given a sequence of error-free training examples (points on Γ) learning corresponds to choosing an 'optimum' $\Gamma_{\mathbf{w}}$ based on the training examples. The property of generalization then corresponds to *interpolation* between the training examples using the surface generated during learning; this interpretation of generalization has also been suggested by Lapedes and Farber [34] and Wolpert [40]. Broomhead and Lowe suggest that the required interpolation should be constructed using the method of *Radial Basis Functions* (RBFs), which has been studied extensively by Powell [41, 42] and others. The standard RBF method performs strict interpolation, that is, it requires that the function constructed perfectly interpolates the training examples. Given a sequence T_k of training examples a function $f_{\mathbf{w}}$ is constructed such that,

$$f_{\mathbf{w}}(\mathbf{x}_i) = o_i \text{ for } i = 1, 2, \ldots, k. \tag{2.27}$$

The function $f_{\mathbf{w}}$ is constructed using a set of basis functions of the form,

$$\phi_i(\mathbf{x}) = \phi\left(\|\mathbf{x} - \mathbf{y}_i\|\right) \tag{2.28}$$

where $\|.\|$ is a norm on $\mathbb{R}^n$, usually the Euclidean norm, the vectors $\mathbf{y}_i \in \mathbb{R}^n$ are called the *centres* of the basis functions and ϕ is a suitable function. Typical choices for the function ϕ are shown in table 2.2; the functions,

$$\phi^*_{MQ}(r) = (r^2 + c^2)^{\beta} \text{ for } 0 < \beta < 1 \text{ and } c \in \mathbb{R}^+ \tag{2.29}$$

$$\phi^*_{IMQ}(r) = (r^2 + c^2)^{-\alpha} \text{ for } \alpha > 0 \text{ and } c \in \mathbb{R}^+ \tag{2.30}$$

are also popular and are clearly more general forms of the multiquadric and inverse multiquadric basis functions. Henceforth, we assume that the inputs $\mathbf{x}_i$ in T_k are distinct. In the most commonly encountered RBF interpolation method the centres correspond to the training example inputs $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$ and $f_{\mathbf{w}}$ is constructed as,

$$f_{\mathbf{w}}(\mathbf{x}) = \sum_{i=1}^{k} \lambda_i \phi\left(\|\mathbf{x} - \mathbf{x}_i\|\right) \tag{2.31}$$

where we use λ_i rather than w_i to denote weights as the former notation is used as standard in the literature. Inserting the constraints of equation 2.27 into equation 2.31 leads to a set of linear equations which can be solved in order to find the weights. Some important results due to Micchelli [43] in fact guarantee that the set of equations has a solution for a large class of functions ϕ and for all k and n provided the training examples are distinct; we discuss this work in detail in chapter 4.

A full definition (see Powell [41] and Poggio and Girosi [44]) of the radial basis function approach requires the construction of $f_{\mathbf{w}}$ as,

$$f_{\mathbf{w}}(\mathbf{x}) = \sum_{i=1}^{k} \lambda_i \phi\left(\|\mathbf{x} - \mathbf{x}_i\|\right) + \sum_{i=1}^{q} \theta_i \psi_i(\mathbf{x}) \text{ where } q \leq k \tag{2.32}$$

where $\mathbf{w}^T = [\ \lambda_1 \quad \lambda_2 \quad \cdots \quad \lambda_k \quad \theta_1 \quad \theta_2 \quad \cdots \quad \theta_q\ ]$ is a vector of weights, $\|.\|$ is the Euclidean norm, $\phi : \mathbb{R}_0^+ \rightarrow \mathbb{R}$ is a continuous basis function and $\{\psi_i \mid i = 1, \ldots, q\}$ is a basis of the linear space $\pi_{d-1}(\mathbb{R}^n)$ of algebraic polynomials from $\mathbb{R}^n$ to $\mathbb{R}$ of degree at most $(d-1)$ for some given d. When using functions of this form in order to interpolate a set of k points the constraints of equation 2.27 give us a set of k linear equations for $(k+q)$ coefficients. We fix the remaining degrees of freedom by requiring that,

$$\sum_{i=1}^{k} \lambda_i \psi_j(\mathbf{x}_i) = 0 \text{ for } j = 1, \ldots, q. \tag{2.33}$$

Again, some important results due to Micchelli [43] are available which allow us to choose ϕ such that it is always possible to interpolate a set of k points, and we discuss these results in detail in chapter 4. Equation 2.31 is a special case of this full definition.

An obvious problem with the radial basis function approach is that a basis function is required for every training example. For many problems, a solution may in fact exist which requires only a small number of basis functions relative to the number of training examples available. In these cases the use of a basis function for every training example can lead to overfitting, that is, in more realistic circumstances when noise is present in the training examples, we may produce a function that fits the noise. In order to avoid this problem the requirement that basis function centres correspond to training examples can be relaxed, either by choosing random elements of the training sequence to use as RBF centres (see also Lowe [29]) or using centres distributed uniformly in the region covered by the data [39]. Also, a constant offset is often introduced into $f_{\mathbf{w}}$ when networks of the form of equation 2.31 are used; the mapping $f_{\mathbf{w}}$ is then,

$$f_{\mathbf{w}}(\mathbf{x}) = \lambda_0 + \sum_{i=1}^{m} \lambda_i \phi\left(\|\mathbf{x} - \mathbf{y}_i\|\right) \text{ where } m < k \tag{2.34}$$

and comparison with equation 2.5 shows that this type of network, which is the type of radial basis function network most often used in practice, is a member of the class of Φ-nets, as are the more general RBFNs of the form of equation 2.32.

A great deal of theoretical work exists which justifies the use of RBFNs. Some theoretical justification for their use is provided by the fact that they were derived from the extensive and rigorously founded theory of multivariable interpolation, which is clearly closely related to the problem of pattern classification. Two further distinct justifications are also available. Firstly, it has been shown by Park and Sandberg [45] that RBFNs can perform universal approximation under quite unrestrictive conditions. Related results have been proved by Cybenko [46] and Hartman *et al.* [47]. Secondly, some RBFNs are a special case of *Regularization Networks*, which we describe below and which have an excellent theoretical basis.

The Choice of ϕ and $\mathbf{y}$

It is reasonable to expect that the choice of radial basis function ϕ and the centre of each basis function will have some effect on the performance of an RBFN. The effect of these choices on learning and generalization ability has been studied by Lowe [29]. In particular, he examines the effect of adapting radial basis functions and their centres using nonlinear optimization techniques as part of the training process. As we noted earlier, this clearly has a detrimental effect in that it usually forces us to optimize on a multimodal error surface and as it increases the computational complexity of the training process. This experimental study leads to the following important conclusions:

1. Any degree of generalization performance achieved using nonlinear optimization can be matched by a radial basis function network using linear optimization (i.e. no adaptation of RBFs or their centres) but using *more* RBFs. Similarly, a particular training performance can be achieved using fewer RBFs if nonlinear optimization is employed. Nonlinear optimization is therefore only necessary if the *smallest possible* network is required.
2. Training takes orders of magnitude longer when nonlinear optimization is used.
3. The precise form of RBF used has little effect on overall performance.

Moody and Darken [30] have introduced a technique for training similar networks of the form of equation 2.35 (below), where $\phi(x) = \exp(-x^2)$, using a combination of linear self-organizing and linear supervised techniques. This allows much of the increase in training time associated with the use of nonlinear optimization to be avoided. Centers $\mathbf{y}_i$ and widths σ_i are obtained using m-means clustering and a p nearest-neighbour heuristic respectively. Weights λ_i are then obtained using the LMS algorithm. This hybrid technique has been extended by Chen *et al.* [48] in applying RBFNs to nonlinear system identification problems, and has also been extended by Musavi *et al.* [49]. The technique in fact appears to have an important consequence in terms of the results obtained in chapter 5 regarding the ability of Φ-nets with fixed basis functions to generalize, and further discussion can therefore be found in subsection 5.2.4.

Related Classifiers

Several networks exist which are closely related to RBFNs. The *Kernel Classifier* (Lippmann [6], Park and Sandberg [45]) computes functions $f_{\mathbf{w}}$ of the form,

$$\begin{aligned} f_{\mathbf{w}}(\mathbf{x}) &= \sum_{i=1}^{m} \lambda_i \phi\left(\frac{\|\mathbf{x}-\mathbf{y}_i\|}{\sigma_i}\right) \\ &= \sum_{i=1}^{m} \lambda_i K\left(\frac{\mathbf{x}-\mathbf{y}_i}{\sigma_i}\right) \qquad (2.35) \end{aligned}$$

where σ_i controls the width of the ith basis function and K is called a *Kernel Function*. Similarly, in the method of *Potential Functions* [4] we use functions $f_{\mathbf{w}}$ of the form,

$$f_{\mathbf{w}}(\mathbf{x}) = \sum_{i=1}^{k} \lambda_i \pi(\mathbf{x}, \mathbf{x}_i). \tag{2.36}$$

The *potential function* $\pi(\mathbf{x}, \mathbf{x}_i)$ is usually maximum for $\mathbf{x} = \mathbf{x}_i$ and decreases monotonically to zero as $\|\mathbf{x} - \mathbf{x}_i\|$ approaches ∞. A typical function is,

$$\pi(\mathbf{x}, \mathbf{x}_i) = \frac{\sigma^2}{\sigma^2 + \|\mathbf{x} - \mathbf{x}_i\|^2}, \sigma \in \mathbb{R}^+. \tag{2.37}$$

2.5.4 Regularization Networks

The similarity between the construction of nonlinear mappings using connectionist networks and the theory of multivariable interpolation has also been exploited by Poggio and Girosi [8, 44, 50, 51, 52, 53], who derive the class of *Regularization Networks* (RNs) using methods from regularization theory.

Again, the learning process is modelled as the task of finding a vector $\mathbf{w}$ which gives the 'best' approximation $f_{\mathbf{w}}$ to a continuous, multivariate function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ on a set of training examples. A question of obvious importance is that of which classes of functions can be approximated by a particular type of network, and one of the main aims of [44] is to develop an approximation method which is general, maps into a multilayer network, and is fully mathematically justifiable. Generalization is again identified with the ability to approximate or interpolate the function between the available training examples, interpolation being regarded as the limit of approximation when the training examples are noise free.

As the training examples are usually noisy, and as they do not in general contain sufficient information to allow unique reconstruction of the function where no training examples are available, *a priori* knowledge in the form of, for example, *smoothness* constraints must be imposed on the mapping[5]. The problem is therefore formulated as a variational problem of finding the function F which minimizes the cost functional,

$$H[F] = \sum_{i=1}^{k} (o_i - F(\mathbf{x}_i))^2 + \lambda \|PF\|^2 \tag{2.38}$$

given a sequence T_k of training examples. In equation 2.38, the first term represents the difference between F and the training examples and the second term is a cost which introduces the required *a priori* information, contained in the constraint operator (stabilizer) P. The parameter $\lambda \in \mathbb{R}_0^+$ is a *regularization parameter* and $\|.\|$ is a norm, usually the L^2 norm (see de Barra [54]), on the function space containing PF. The regularization parameter controls the compromise between quality of approximation and smoothness (or other *a priori* constraint) and hence can be thought of as controlling the way in which generalization is enforced. An example of a stabilizer is,

$$\|O^j F\|^2 = \sum_{i_1=1}^{n} \sum_{i_2=1}^{n} \cdots \sum_{i_j=1}^{n} \int_{\mathbb{R}^n} d\mathbf{x} (\partial_{i_1 \cdots i_j} F(\mathbf{x}))^2 \tag{2.39}$$

where $j \geq 1$ and $\partial_{i_1 \cdots i_j} = \partial^j / \partial x_{i_1} \cdots \partial x_{i_j}$, which leads to *multidimensional splines*; other examples can be found in [44]. The overall method can also be theoretically justified using the theory

[5]The use of smoothness constraints has also been suggested by Barron and Barron [10].

of Bayesian estimation, see [44, 51]. Minimizing H gives a solution,

$$f_{\mathbf{w}}(\mathbf{x}) = F(\mathbf{x}) = \sum_{i=1}^{k} w_i G(\mathbf{x}; \mathbf{x}_i) \tag{2.40}$$

where $G(\mathbf{x}; \mathbf{y})$ is the Green's function (see [44]) of $\hat{P}P$, $\hat{P}$ is the adjoint of P, and the weights w_i satisfy,

$$(\mathbf{G} + \lambda\mathbf{I})\mathbf{w} = \mathbf{o} \tag{2.41}$$

where the matrix $\mathbf{G}$ is defined by $G_{ij} = G(\mathbf{x}_i; \mathbf{x}_j)$. A full derivation of this result is given in [44, 51] along with a full discussion regarding the existence of a unique solution to 2.41, and the case of pure interpolation ($\lambda = 0$). Note that it may be necessary to add a further, usually polynomial, term to the right-hand side of equation 2.40; this depends on the stabilizer used and the extra term is usually omitted (see [50] for a discussion). Comparison of equation 2.40 with equation 2.5 shows that regularization networks are Φ-nets, even if a further polynomial term is added.

An important special case appears when P is rotationally and translationally invariant. In this case G is a radial function,

$$G = G(\|\mathbf{x} - \mathbf{y}\|) \tag{2.42}$$

and hence the class of regularization networks contains some of the RBFNs (not all RBFNs are regularization networks). Rotational and translational invariance of P is in fact quite common in practice.

Several modifications to the regularization network approach are discussed in [44, 50, 51, 52, 53], including the use of fewer centres than training examples and the adaptation of centres. In particular, the pseudoinverse based approximation method used by Broomhead and Lowe [39] is obtained as a special case.

2.5.5 The Modified Kanerva Model

The *modified Kanerva model* (MKM) was introduced by Prager and Fallside [55] as an extension of the *Kanerva Memory Model* introduced by Kanerva [56]. The main difference, for our purposes, between the original and modified Kanerva models is that the latter is able to operate with real-valued, rather than only binary input patterns.

The MKM uses basis functions of the form,

$$\phi_i(\mathbf{x}) = \begin{cases} 1 & \text{if } \mu_p(\mathbf{x}, \mathbf{y}_i) < r \\ 0 & \text{otherwise} \end{cases} \tag{2.43}$$

where $\mathbf{y}_i \in \mathbb{R}^n$ is a *location* associated with basis function ϕ_i, $r \in \mathbb{R}^+$ and p, which is a positive integer or ∞, are fixed parameters, and μ_p is a distance metric, generally of the form,

$$\mu_p(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n} | x_i - y_i |^p \right)^{\frac{1}{p}}. \tag{2.44}$$

Each basis function thus produces an output of 1 (is *active*) if the current input falls within a hyperball centred on $\mathbf{y}_i$, the actual shape of which depends on the metric μ_p used. The output of the basis function is 0 otherwise. Typical choices of p (Clarke *et al.* [57]) are $p = 2$, leading to hyperspherical regions, $p = \infty$, leading to hypercubes, and $p = 1$, leading to the duals of hypercubes. The shape of the hyperball used is generally chosen such that it closely matches the shape of the region populated with locations [55], although $p = 2$ is generally avoided as it is less convenient to implement, and an analysis in [57] has shown that the use of μ_∞ leads to a sparsely connected first layer which is highly desirable as it leads to a significant increase in the speed with which the network can be simulated.

The locations are fixed and are typically chosen randomly; suggestions are given in [57] regarding the introduction of adaptation to this layer and Prager [58] introduces an effective method for pruning unnecessary locations. Inputs to the network are constrained to lie within a relatively small region in the centre of the region populated by locations. This allows us to ensure that approximately the same fraction of the total number of basis functions is active for any input; the actual fraction which becomes active is set, typically to $\frac{1}{10}$ (see [55, 57]), by fixing a suitable value for the *activation radius* r. In some cases the weights are constrained to be integer valued [57].

In Kanerva's original work he predicted that the most significant properties of his model would only be exhibited by networks much larger than could be simulated using serial computers. Although in [55] Prager and Fallside use networks small enough such that many of the most important properties predicted theoretically by Kanerva do not appear, and as a result the networks are less successful than Kanerva's work would suggest, they do show that the MKM is as powerful as some alternative models in the literature when applied to a particular speech recognition problem.

2.5.6 Other Φ-Nets

Clearly, we can construct a Φ-net using any appropriate set of basis functions. The networks presented above are those most often presented in the literature, and an obvious way in which to construct further Φ-nets would be to mix basis functions from these networks.

A further Φ-net of interest can be constructed using basis functions of the form of equation 2.15, that is, of the form of the processors in an MLP. In this case we would regard parameters associated with each basis function, including any steepness parameter β associated with the activation function σ, as being fixed rather than adapting, and would obtain a network similar to a single hidden layer MLP. It is important to note that when we discuss MLPs in this dissertation we mean to refer to MLPs in the generally understood sense, in which all parameters (with the usual exception of the steepness parameters) are adapted.

Some further standard network types are also specific examples of Φ-nets. For example, the *cerebellar model articulation controller (CMAC)*, which was introduced by Albus [59] and is reviewed by Miller *et al.* [60], and the *Distributed Method*[6] (Gallant and Smith [61]) which uses basis functions that are linear discriminants with randomly chosen coefficients.

2.6 Discussion

Φ-nets with fixed basis functions have, in recent years, been marginalized by many researchers. We believe that in most cases the arguments advanced against these networks are fallacious, and the purpose of this section is to show why this is the case, and to argue that Φ-nets with fixed basis functions are useful pattern classifiers which are a viable alternative to MLP type networks. We begin with a brief attempt to explain why these networks have been marginalized.

[6]This was brought to the attention of the author by Dr. S. Renals.

2.6.1 Φ-Networks: A Short Historical Critique

As in any other discipline, it is important in pursuing research in connectionist networks to appreciate their history, perhaps especially so in this particular case as the pursuit is highly interdisciplinary. As we have already observed, networks of the general form of Φ-nets have been studied since the 1960s. Originally, PDFs attracted particular attention as a natural extension of LDFs which were not subject to the same well-known limitations, and at this time a great deal of high quality research was produced. Research in connectionist networks then suffered a well-publicized lull after the publication in 1969 of Minsky and Papert's results [3], and when interest was re-kindled with the introduction by Rumelhart *et al.* [19] of the HLBP algorithm, much of the foregoing research was forgotten in the rush to join the MLP bandwagon. In retrospect, it is surprising that the HLBP algorithm caused such a sudden and significant burst of enthusiasm among researchers, as, contrary to the perceived state of affairs, good alternative algorithms for estimating multilayer networks had already been available for some time (Barron and Barron [10]). Clearly, this was merely a consequence of unfamiliarity with the early literature.

It is also surprising that even now the MLP trained using an HLBP type algorithm tends to be the immediate choice in practical experimental studies, regardless of its well-known drawbacks. Minsky and Papert [3] suggest that it was a failure to fully appreciate the potential scope and importance of their results, and a failure to realize that they would be likely to apply to networks other than standard perceptrons, as well as an incomplete understanding of the problems associated with hill-climbing methods, which led to the prevalent high degree of interest in MLPs. We are of the opinion that, in a similar manner, it is an incomplete knowledge of the existing results, and an incomplete understanding of the issues involved, which has led to the general lack of enthusiasm for Φ-nets with fixed basis functions.

During the 1980s, much of the early work on specific examples of Φ-nets was rediscovered. Rayner and Lynch [28, 62], working from the point of view of nonlinear adaptive filter theory, rediscovered the idea of using PDFs and similar ideas were re-investigated by Chen *et al.* [33] and by Maxwell, Giles and Lee [37, 38]. Rumelhart *et al.* [19] suggested that feedforward networks using this type of unit, under the name of *Sigma-Pi units*, could be used as an alternative to multilayer perceptrons of the usual type, although they did not realize that such units could be useful in their own right without combining them to form larger networks and hence introducing the standard problems associated with training multilayer networks. Radial basis function networks, regularization networks, the modified Kanerva model and the distributed method were also introduced during the 1980s, although it has not been generally appreciated that these are all members of the class of Φ-nets originally studied in some form in the 1960s. What has been provided by recent research, particularly in PDFs, RBFNs and RNs, are further rigorous mathematical justifications for the use of Φ-nets with particular sets of basis functions. What also appears consistently in recent research, which we summarize below, is that Φ-nets with fixed basis functions can often be made to provide performance broadly equivalent to that of MLPs, with decreases of orders of magnitude in training time, when applied to practical problems.

It is clear that some Φ-nets have disadvantages when applied to some problems — for example PDFs are not necessarily a good solution to time series prediction problems as explained above. However, in many cases they are a good alternative to multilayer perceptrons, as we will now attempt to demonstrate.

2.6.2 Examples of the Practical Application of Φ-Nets

In order to further motivate the use of Φ-nets we now provide a brief summary of some practical problems to which they have been successfully applied. The examples provided apply to networks with fixed basis functions. Note however that in some cases a simple algorithm was used to select a 'sensible' set of basis functions; for example, one of the simple strategies mentioned in our review of RBFNs. The important point to remember here is that in these cases the algorithms used to select basis functions are very simple and very fast, and so we retain the advantage of fast training which motivated us to study these networks.

Polynomial Discriminant Functions

Rayner and Lynch [28, 62, 63] have compared the performance of PDFs with that of multilayer perceptrons for the parity problem and for the recognition of both character font and hand drawn characters, obtaining comparable performance with significant reductions in training time.

Chen *et al.* [33] have used a similar type of network in order to perform channel equalisation and compared its performance with that of a multilayer perceptron, finding that although very similar results were obtained, PDFs were much easier to train. They show that when using a network as described by equation 2.24 it may be necessary to use a large d, and hence a large number of weights, but that this problem may be solved by using,

$$f_{\mathbf{w}}(\mathbf{x}) = \rho(g_s(\mathbf{w}^T\tilde{\mathbf{x}})) \tag{2.45}$$

where g_s is a sigmoid,

$$g_s(x) = \tanh\left(\frac{\alpha x}{2}\right), \alpha > 0. \tag{2.46}$$

Kreßel *et al.* [64] have compared the performance of a PDF with that of a multilayer perceptron having a comparable number of weights in solving a handwritten digit recognition problem, obtaining comparable performance. Finally, studies by Rajan and Rayner [65] using PDFs in the classification of rock types using acoustic signals from the drill bit in an oil well have yielded good results.

Radial Basis Function Networks

Broomhead and Lowe [39] have successfully applied RBFNs to chaotic time series prediction, and Niranjan and Fallside [66] compare multilayer perceptrons, RBFNs and MKMs for the recognition of static speech patterns, finding that they provide comparable performance. Lowe [29] successfully applies RBFNs to vowel classification, and Renals and Rohwer [12] apply RBFNs to phoneme classification problems, again obtaining performance similar to that of a multilayer perceptron. Lee [67] obtains good results in applying RBFNs to handwritten digit recognition.

Modified Kanerva Models

Prager and Fallside [55] obtain good results in speech recognition using an MKM. Similarly encouraging results were obtained by Clarke *et al.* [57]. A more recent study by Prager [58] addresses vowel classification, word classification and wheat classification (detection of one particular wheat strain from among 22 others) and again obtains good results.

Finally, Boser *et al.* [11] have successfully applied various Φ-nets to problems involving the recognition of handwritten characters.

2.6.3 Possible Criticisms of Φ-Nets

We now summarize and discuss some possible (and in some cases commonly aired) objections to the use of Φ-nets, in particular those with fixed basis functions.

Constructing an Extension

A standard objection to Φ-nets with fixed basis functions is that it appears that the actual basis functions used must be selected in an arbitrary manner. A related objection is that the use of a specific set of basis functions imposes a particular form on the class of functions which the network can compute. In answer to this criticism, we first note that universal approximation results similar to those available for MLPs exist for various types of basis function; some of these were mentioned above. However, although results of this type are theoretically appealing, they tend not to be very useful in any practical sense. We therefore also note that excellent theoretical justification exists for the use of many types of basis function, particularly for RBFNs and RNs; the same cannot be said for the MLP in which the use of simple nodes containing a weighted summation followed by a nonlinearity is based on a highly idealized biological approximation.

In answer to the criticisms regarding the biasing of the class of functions, note that the same criticism can be applied to *any* connectionist network, including the MLP. What is important is that the class of functions computed is general enough to cover the types of problem we actually expect to encounter, that is, to allow us to approximate the types of mapping that are likely to arise in practice.

The Requirement of Full Connection

A further standard criticism is that some Φ-nets must be fully connected[7]. Examples are RBFNs, RNs, and to some extent MKMs although as we have seen the use of a suitable distance metric leads to a sparsely connected first layer. This means that m different vectors $\mathbf{y}_i \in \mathbb{R}^n$, one specifying the parameters for each basis function, must be stored in addition to the weight vector $\mathbf{w}$. In some cases, such as Kernel Classifiers, further parameters must also be stored. The criticism here is that for large m and n this leads to an unrealistically large storage requirement. We do not agree with this criticism in general as in a realistic practical sense the on-going improvements in memory technology and custom connectionist network integrated circuit design make this unlikely to be a significant problem. Note also that again the same criticism can be applied to *any* fully connected feedforward connectionist network.

A related criticism is that the increased complexity of many standard basis functions over the complexity of the nodes typically used in MLPs leads to an unacceptable increase in the time taken to classify a new input vector. Again, we expect that this should not, considering the current, celebrated rate of technological advance, be a significant problem in all but cases with the most stringent speed requirements.

The Size of the Weights

One important observation made in [3] is that we must be careful to consider the size of the weights required in solving a problem. It is tempting to ask whether the ease of training as-

[7]Φ-nets clearly have a structure generally described as having a single hidden layer, in which basis functions are 'hidden nodes'. Connections from hidden nodes to the output node each have a single weight. In some Φ-nets, such as radial basis function networks, a fixed or variable weight must be associated with each connection from an input to a basis function, and all such connections may be included.

sociated with Φ-nets with fixed basis functions is obtained at the cost of introducing the risk of requiring impractically large weights. We have not studied this question in detail. Several relevant theoretical results exist (see for example Muroga [68] and Saks [69]), although we are not aware of any comprehensive practical study addressing this question which consequently provides an important area for further research. We note that in practice it does not in general appear to have been a significant drawback. Renals and Rohwer [12] report that in the specific cases that they addressed experimentally it caused problems up to about 10% of the time in RBFNs having more than 150 centres, but that methods could be introduced to improve the situation. Niranjan and Fallside [66] experienced some problems of this type using MKMs, but their experience does not appear to have been shared by others using these networks.

The Size of the Network

It is generally perceived that in using Φ-nets with fixed basis functions it will be necessary to use a larger network to solve a problem than if an MLP is used. This criticism is most often applied to PDFs; see the discussion in subsection 2.5.2.

The precise validity of this criticism is not clear. Although we can in fact expect it to be true in general for reasons which we will state and discuss in chapter 4, it is important to appreciate that it depends on the specific problem addressed and the set of basis functions used — some problems can be solved by either a small Φ-net or a larger MLP. It is also important to appreciate that for many standard basis functions the increase in the size of the network required is nowhere near as bad as may be expected after an examination of the rate at which the number of weights in a PDF can increase; this statement is supported by many of the experimental results provided above and also by the fact that, given T_k, a radial basis function network can be constructed which has k basis functions with fixed centres and which can exactly learn the training examples (see chapter 4).

What About Minsky and Papert's Objections?

Φ-nets are obviously rather similar to the perceptrons studied by Minsky and Papert [3]. An obvious difference is that the former have inputs in $\mathbb{R}^n$ and basis functions $\phi_i : \mathbb{R}^n \to \mathbb{R}$, whereas the latter have inputs in $\{0,1\}^n$ and partial predicates $\phi_i : \{0,1\}^n \to \{0,1\}$; this difference is relatively superficial, and it is natural to ask whether Minsky and Papert's negative results apply to Φ-nets. The important point to remember here is that many of Minsky and Papert's results rely on the fact that partial predicates have been limited in some manner; as a result of the fact that we allow Φ-nets to be fully connected we are effectively deciding not to impose such restrictions. Note however that neither LDFs or PDFs are necessarily fully connected.

2.7 Conclusion

In this chapter we have introduced and reviewed in detail the class of networks with which this dissertation is concerned. Our main motivation for studying networks of this type was that they offer significant advantages in terms of training time when compared to the usual feedforward network of choice: the multilayer perceptron. The class of networks that we consider is similar to one introduced in the 1960s; many well-known standard connectionist networks are specific examples of this class, although this has not been widely appreciated, and we have reviewed many of them in detail. We have also briefly reviewed the available training algorithms for this class of networks. Finally, we have further motivated the use of this class of networks by reviewing some practical applications in which they have been used successfully, and we have discussed some possible criticisms of the networks, in most cases arguing that they are in general unlikely to present significant problems.

Part II

Generalization

Chapter 3

Computational Learning Theory and the Theory of Generalization

3.1 Introduction

The first of our aims in this dissertation is to investigate the ability of Φ-nets to generalize. In this chapter we introduce the theoretical framework which will be used to achieve this aim, and we briefly review the alternative formalisms available for the theoretical analysis of generalization.

We have chosen to use methods from computational learning theory in our analysis. Although the body of work on theoretical techniques for the analysis of generalization forms only a relatively small part of the overall published literature on connectionist networks, it is nonetheless composed of several approaches based on distinct underlying frameworks. Only two of these approaches have made a significant impact to date: the approach based on computational learning theory and the approach based on statistical physics. The former approach is attractive for two reasons. Firstly, its results are arguably the more powerful because they tell us about performance in the *worst case*, rather than the *average case*; however, as we shall see, they are also in a sense quite limited at present. Secondly, the generalization performance of Φ-nets with fixed basis functions has not previously been addressed using this approach, while on the other hand, results obtained using this approach are available for the generalization performance of networks other than Φ-nets, providing a basis for comparison.

In section 3.2 we provide an introduction to the relevant computational learning theory, which is used in subsequent chapters. The theory is in fact very general and applies to systems other than connectionist networks. However, as connectionist networks are our primary concern this introduction is biased towards their treatment. In section 3.3 we briefly review the alternative approaches to the theoretical analysis of generalization, and in section 3.4 we briefly discuss the importance of the *capacity* of a network in considering its generalization performance. Section 3.5 concludes the chapter.

3.2 Computational Learning Theory

3.2.1 The Hypothesis Space

In analysing generalization, we will consider networks applied to pattern classification problems having two classes, and we therefore consider threshold Φ-nets which compute functions $f_{\mathbf{w}}(\mathbf{x})$:

$\mathbb{R}^n \to \mathbb{B}$ and have an associated class $\mathcal{F}_n^\Phi$ of functions as defined in chapter 2. These networks classify a pattern $\mathbf{x} \in \mathbb{R}^n$ to be in class 1 if $f_\mathbf{w}(\mathbf{x}) = +1$ and to be in class 2 if $f_\mathbf{w}(\mathbf{x}) = -1$. With each $f_\mathbf{w} \in \mathcal{F}_n^\Phi$ we can associate a subset $h_\mathbf{w}$ of $\mathbb{R}^n$ called an *hypothesis*. The set of all hypotheses forms the *hypothesis space*.

Definition 3.1 (Hypothesis space) *Consider a threshold Φ-net which computes a class of functions $\mathcal{F}_n^\Phi$. Given a particular weight vector* $\mathbf{w}$ *the network computes a specific function $f_\mathbf{w} : \mathbb{R}^n \to \mathbb{B}$. The* hypothesis $h_\mathbf{w}$ *associated with $f_\mathbf{w}$ is the subset of $\mathbb{R}^n$ defined as,*

$$h_\mathbf{w} = \{\mathbf{x} \in \mathbb{R}^n \mid f_\mathbf{w}(\mathbf{x}) = +1\}, \tag{3.1}$$

that is, the region of $\mathbb{R}^n$ for which the output of the network is $+1$. The hypothesis space H *of the network is the set of all hypotheses,*

$$H = \left\{h_\mathbf{w} \mid \mathbf{w} \in \mathbb{R}^{m+1}\right\}. \tag{3.2}$$

3.2.2 The Growth Function and the VC Dimension

It should be clear that definition 3.1 is not exclusive to threshold Φ-nets, but applies to *any* system which can be modelled by an appropriate parameterized class of functions. The same is true of all the computational learning theory presented in this section.

Two combinatorial parameters associated with $\mathcal{F}_n^\Phi$ and H have proved to be of central importance to computational learning theory; they are the *growth function* and the *Vapnik-Chervonenkis (VC) Dimension.* These parameters were originally introduced by Vapnik and Chervonenkis [70] in their study of the uniform convergence of relative frequencies to probabilities. The VC dimension can be regarded as a generalization of Cover's concept of *capacity* [17] — it is a measure of the 'expressive power' of $\mathcal{F}_n^\Phi$ and H (Anthony and Biggs [71]). In fact, some measure of the capacity of a network tends to be important whenever we consider the property of generalization in feedforward networks, regardless of the particular formalism used, and we discuss this further in section 3.4.

Consider a threshold Φ-net which computes the class $\mathcal{F}_n^\Phi$ of functions and has hypothesis space H, and let $S_k \subseteq X$ be a set of k elements of the *environment* X, which corresponds to the input space of the network and in this dissertation is always $\mathbb{R}^n$ unless otherwise stated.

Definition 3.2 (Dichotomy) *Given S_k and a function $f_\mathbf{w} \in \mathcal{F}_n^\Phi$ we define the* dichotomy *(sometimes called the* two-colouring*) (S^+, S^-) of S_k induced by $f_\mathbf{w}$ as the partition of S_k into the two disjoint subsets S^+ and S^-, where $S^+ \cup S^- = S_k$, in such a way that for $\mathbf{x} \in S_k$, $\mathbf{x} \in S^+$ if $f_\mathbf{w}(\mathbf{x}) = +1$ and $\mathbf{x} \in S^-$ otherwise.*

Definition 3.3 *Given the hypothesis space H and $S_k \subseteq X$, we define the set $\triangle_H(S_k)$ as*[1]*,*

$$\triangle_H(S_k) = \{h \cap S_k \mid h \in H\}. \tag{3.3}$$

If $\triangle_H(S_k) = 2^{S_k}$, where 2^{S_k} denotes the set of all subsets of S_k, then we say that S_k is shattered *by H.*

The set $\triangle_H(S_k)$ therefore contains as elements all the sets S^+ which can be induced by the functions in $\mathcal{F}_n^\Phi$. Figure 3.1 illustrates the idea of an hypothesis space, a dichotomy, and the set

[1]In this and the next definition, some authors use the symbol Π instead of $\triangle$.

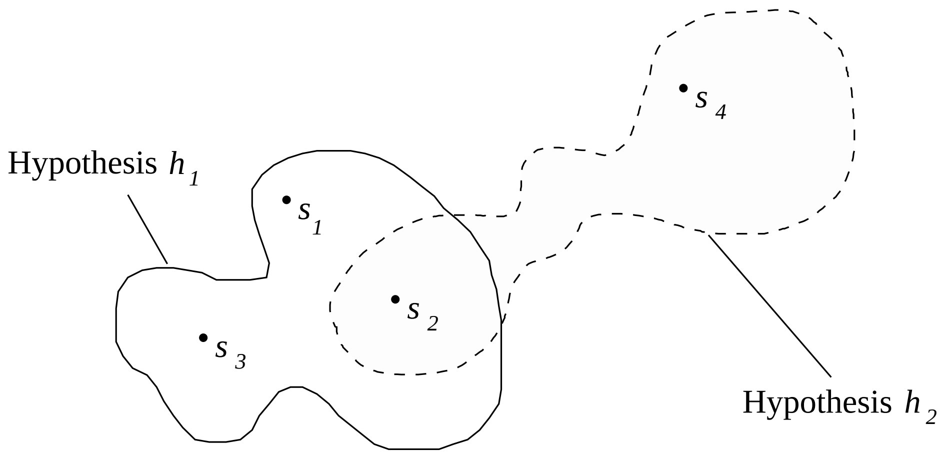


Figure 3.1: Illustration of the ideas of an hypothesis space, a dichotomy, and the set $\triangle_H(S_k)$ for an environment $X = \mathbb{R}^2$ and $k = 4$, using a set of points $S_4 = \{s_1, s_2, s_3, s_4\}$ and an hypothesis space $H = \{h_1, h_2\}$.

$\triangle_H(S_k)$ for a simple example. In this example we let $X = \mathbb{R}^2$ and consider the set of points $S_4 = \{s_1, s_2, s_3, s_4\}$ and the hypothesis space $H = \{h_1, h_2\}$. Using the hypotheses h_1 and h_2 illustrated we have,

$$\triangle_H(S_4) = \{\{s_2, s_4\}, \{s_1, s_2, s_3\}\}. \tag{3.4}$$

If we consider the functions f_1 and f_2 corresponding to h_1 and h_2 respectively, f_1 induces the dichotomy,

$$d_1 = (S^+ = \{s_1, s_2, s_3\}, S^- = \{s_4\}) \tag{3.5}$$

and f_2 induces the dichotomy,

$$d_2 = (S^+ = \{s_2, s_4\}, S^- = \{s_1, s_3\}). \tag{3.6}$$

We can now define the growth function and the VC dimension.

Definition 3.4 (Growth function) *We define the* growth function $\triangle_H(i)$ *on the set of positive integers as,*

$$\triangle_H(i) = \max_{S_i \subseteq X} (\mid \triangle_H(S_i) \mid). \tag{3.7}$$

The growth function therefore tells us the *maximum* number of distinct dichotomies induced by $\mathcal{F}_n^\Phi$ for *any* set of i points.

Definition 3.5 (Vapnik-Chervonenkis dimension) *Given the hypothesis space H, we define the* Vapnik-Chervonenkis dimension $\mathcal{V}(H)$ *of H as the largest integer i such that $\triangle_H(i) = 2^i$. If no such i exists then $\mathcal{V}(H)$ is infinite.*

The VC dimension therefore tells us the largest i such that a set of i points exists which is shattered by H. It is easy to show that if H is finite, then $\mathcal{V}(H) \leq \log_2 \mid H \mid$, however in this dissertation we will usually be concerned with infinite hypothesis spaces. Note that as an hypothesis space H is directly related to a class of functions $\mathcal{F}$, and to a corresponding connectionist network, we can refer to the growth function and the VC dimension of $\mathcal{F}$, or of the connectionist network, and use the quantities $\triangle_{\mathcal{F}}(S_k)$, $\triangle_{\mathcal{F}}(i)$ and $\mathcal{V}(\mathcal{F})$ defined in the obvious manner.

There are no known systematic methods for calculating either the growth function or the VC dimension, and in general we must therefore exercise our own ingenuity in order to calculate them for specific cases; the next chapter of this dissertation addresses precisely this problem for the case of Φ-nets. The VC dimension tends to be the easier of the two parameters to deal with, and luckily if we know the value of the VC dimension then we can bound the growth function using a result commonly known as *Sauer's lemma* [72], and an extension due to Blumer *et al.* [73] (see [71] for further references regarding this result).

Lemma 3.6 (Sauer's Lemma) *Given a class of functions $\mathcal{F}$ for which $\mathcal{V}(\mathcal{F}) = d \geq 0$ and $d < \infty$,*

$$\triangle_{\mathcal{F}}(k) \leq \Theta(d,k) = 1 + \sum_{i=1}^{d} \binom{k}{i}, \tag{3.8}$$

where $k \geq 1$ is a positive integer. When $k \geq d \geq 1$,

$$\Theta(d,k) < \left(\frac{ek}{d}\right)^{d}. \tag{3.9}$$

This lemma is useful as it is often easier to estimate $\mathcal{V}(\mathcal{F})$ than $\triangle_{\mathcal{F}}(k)$, and as it gives an upper bound on $\triangle_{\mathcal{F}}(k)$ which is a polynomial function of k of degree d. A further result [70, 71] on $\triangle_{\mathcal{F}}(k)$ for finite $\mathcal{V}(\mathcal{F})$ is that either $\triangle_{\mathcal{F}}(k) = 2^k$ or,

$$\triangle_{\mathcal{F}}(k) \leq k^{\mathcal{V}(\mathcal{F})}. \tag{3.10}$$

Clearly when $\mathcal{V}(\mathcal{F})$ is infinite, $\triangle_{\mathcal{F}}(k) = 2^k$ for all k. The bound of equation 3.10 is important for reasons which we will provide below.

Values for the growth function and VC dimension, or bounds thereon, are known for a number of hypothesis spaces, some of which correspond to connectionist networks. Rather than list them here we introduce them as they become relevant.

3.2.3 Using the Growth Function and VC Dimension to Analyse Generalization

Consider a connectionist network which computes a class $\mathcal{F}$ of functions. One way of interpreting the task of training this network is as follows: we attempt to adjust the weight vector $\mathbf{w}$ in order to obtain a function $f_{\mathbf{w}} \in \mathcal{F}$, which gives 'good agreement' with a *target function* f_T on a set of training examples. Let $\mathbf{x}$ be an element of the environment X picked at random according to some arbitrary[2] distribution P on X. We define $\pi_{f_{\mathbf{w}}}$ as the probability that $f_{\mathbf{w}}$ agrees with f_T on an example chosen at random according to P, that is,

$$\pi_{f_{\mathbf{w}}} = \Pr\left[f_{\mathbf{w}}(\mathbf{x}) = f_T(\mathbf{x})\right]. \tag{3.11}$$

Given a sequence $T_k = ((\mathbf{x}_1, f_T(\mathbf{x}_1)), \ldots, (\mathbf{x}_k, f_T(\mathbf{x}_k)))$ of training examples in which the $\mathbf{x}_i$ are picked at random according to P, we define $v_{f_{\mathbf{w}}}$ as the fraction of the k examples which $f_{\mathbf{w}}$

[2]The theory presented in this chapter is distribution independent. It is important to remember however that, while we can use an entirely arbitrary distribution to model the generation of data, in any specific situation the *same* distribution is used in both the training of the network and in the assessment of its subsequent performance.

classifies correctly. When we train a connectionist network we choose a particular $\mathbf{w}$ on the basis of the value of $v_{f_\mathbf{w}}$, and it is therefore important to know whether $v_{f_\mathbf{w}}$ converges to $\pi_{f_\mathbf{w}}$ in a uniform manner for all $f_\mathbf{w} \in \mathcal{F}$ as k becomes large. If this is not the case then it is possible to choose a function $f_\mathbf{w}$ for which the corresponding value of $\pi_{f_\mathbf{w}}$ is in fact relatively low. An important inequality due to Vapnik and Chervonenkis [70] bounds the probability that there is some $f_\mathbf{w} \in \mathcal{F}$ for which $v_{f_\mathbf{w}}$ and $\pi_{f_\mathbf{w}}$ differ significantly. Given a particular value ϵ, we have,

$$\Pr\left[\sup_{f_\mathbf{w}\in\mathcal{F}} \mid v_{f_\mathbf{w}} - \pi_{f_\mathbf{w}} \mid > \epsilon\right] \leq 4\triangle_\mathcal{F}(2k)\exp\left(\frac{-\epsilon^2 k}{8}\right). \tag{3.12}$$

This result has recently been improved by Anthony and Shawe-Taylor [74], and various similar results have been proved by other authors (see Saitta and Bergadano [75]).

An important consequence of equation 3.10 is now apparent. When $\mathcal{V}(\mathcal{F})$ is finite the growth function $\triangle_\mathcal{F}(k)$ is bounded above by a polynomial function of k. As the factor $\exp\left(\frac{-\epsilon^2 k}{8}\right)$ decays exponentially in k it is therefore possible, by choosing k large enough, to make the right-hand side of equation 3.12 arbitrarily small. Equation 3.12 places a bound on the rate of convergence which is independent of both the specific target function f_T and the specific probability distribution P. The speed of convergence, and hence the number of examples required to guarantee a particular generalization performance, is clearly influenced by $\mathcal{V}(\mathcal{F})$.

Further discussion of this method of analysing generalization performance can be found in Hertz *et al.* [15] and Abu-Mostafa [76] (on which the preceding discussion in this subsection is based).

3.2.4 Generalization and PAC Learning Theory

The discussion given above illustrates one reason for the importance of the growth function and VC dimension in the analysis of generalization in connectionist networks. The work on generalization presented in this dissertation is based on the theory of *Probably Approximately Correct (PAC) learning* introduced by Valiant [77]. In particular, in chapter 5 we use an extended form of PAC learning due to Blumer *et al.* [73], based on the work of Vapnik [78], to analyse the generalization performance of Φ-nets. The same collection of techniques was used by Baum and Haussler [79] in their well-known work in which they address the generalization ability of general feedforward networks constructed using linear threshold elements; we define these networks in full in the next chapter.

The published literature in this field is again too extensive to allow an exhaustive review, and we therefore give only the minimum introduction required as a prerequisite for the work presented in chapter 5. In particular, note that in general PAC learning theory one also addresses the question of whether learning algorithms are computationally efficient, although we will not be concerned specifically with this type of question here. More detailed introductions to standard PAC learning can be found in Natarajan [80], Anthony and Biggs [71], Angluin [81], Kearns [82], and Haussler [83] and further material on the extended form in Blumer *et al.* [73], on which the material in the remainder of this subsection is based.

Standard PAC Learning

Consider again the environment X corresponding to the input space of a network, and an hypothesis space H computed by the network. We define the *concept class* C, in a similar manner

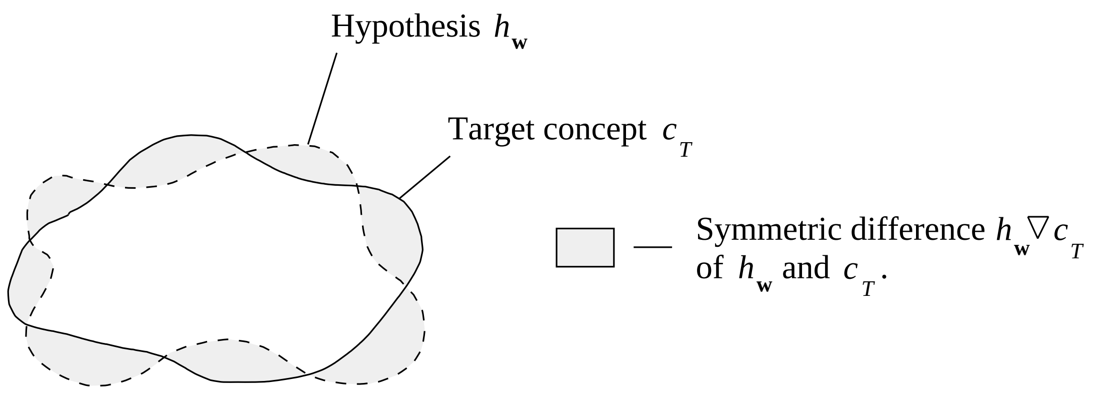

Hypothesis $h_{\mathbf{w}}$
Target concept c_T
▨ — Symmetric difference $h_{\mathbf{w}}\nabla c_T$ of $h_{\mathbf{w}}$ and c_T.

Figure 3.2: The error of an hypothesis $h_{\mathbf{w}}$ is the probability according to P of the symmetric difference $h_{\mathbf{w}}\nabla c_T$.

to H, as a set of subsets of X and require that elements of C and H are Borel sets[3]. The concept class C may or may not be equal to H. Training the network corresponds to adjusting the weight vector $\mathbf{w}$ such that the network computes an hypothesis $h_{\mathbf{w}} \in H$ which is a 'good approximation' to some *target concept* $c_T \in C$.

In PAC learning, we are given a finite *sample* $T_k = ((\mathbf{x}_1, o_1), (\mathbf{x}_2, o_2), \ldots, (\mathbf{x}_k, o_k))$ of k training examples, where inputs $\mathbf{x}_i$ are drawn independently according to some arbitrary fixed distribution P on X and labelled with output $o_i = +1$ if $\mathbf{x}_i \in c_T$ and $o_i = -1$ otherwise. Let X_k denote the corresponding k-tuple $X_k = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k)$. We call a function F which, given a large enough sample of any $c_T \in C$, returns an $h_{\mathbf{w}} = F(T_k) \in H$ which with *high probability* is a *good approximation* to c_T, a *learning function* for C with respect to P. In order to formalize this we define the *error* of some $h_{\mathbf{w}} \in H$ as the probability according to P of $h_{\mathbf{w}}\nabla c_T$, where $a\nabla b$ is the symmetric difference[4] of the sets a and b. The error is thus the probability that c_T and $h_{\mathbf{w}}$ disagree on a randomly drawn example; this is illustrated in figure 3.2. We then require that, given some small specified ϵ and δ, the hypothesis $h_{\mathbf{w}}$ provided by the learning function satisfies,

$$\Pr\left[\text{Error of } h_{\mathbf{w}} > \epsilon\right] \leq \delta \tag{3.13}$$

uniformly for all $c_T \in C$ provided k is large enough. Equation 3.13 is rather informal, and is intended to provide an intuitive idea of what is required. Formally, we require,

$$P^k\left[\{X_k \mid P\left[F(T_k)\nabla c_T\right] > \epsilon\}\right] \leq \delta \tag{3.14}$$

where k must depend *only* on ϵ and δ and where the notation $P[E]$ denotes the probability of the event E according to the distribution P. If this is possible regardless of the actual distribution P then we say that F is a learning function for C with sample size k. The smallest k guaranteed to achieve this is called the *sample complexity* of F, and any concept class for which such a learning function exists is called *uniformly learnable by H*.

[3]We also require that C satisfies some measurability-related conditions. Neither of these requirements is very restrictive and neither presents a problem in practice. A discussion is given an appendix A where, in particular, we argue that all elements of C and H are Borel sets in realistic situations, and prove that *all* classes computed by Φ-nets satisfy the required measurability-related conditions.

[4]The symmetric difference of two sets is the set of elements which belong to one, but not both, of the sets. Formally, $a\nabla b = (a \setminus b) \cup (b \setminus a)$. It is more common to use the notation $a\triangle b$, however we use this notation in order to avoid confusion with the growth function.

Blumer *et al.* [73, 84] have shown that the VC dimension is an extremely important parameter in this type of learnability theory. For example, they prove that a concept class C is uniformly learnable by some hypothesis space H if and only if $\mathcal{V}(C)$ is finite. But how are these ideas related to generalization? Note that from equation 3.14 we can require that the error of the hypothesis computed by a network *after* training is, with high probability, below some arbitrary level. This error describes the performance of the network for *all possible inputs*, and not just those used during training, and hence provides a direct measure of generalization performance.

Extended PAC Learning

PAC learning in its standard formulation does not provide a satisfactory means of dealing with several important situations. For example, it does not allow us to consider samples T_k which contain misclassifications and, perhaps most importantly from our point of view, it does not allow us to consider the use of a target concept defined in a stochastic manner — an assumption which is often used in work on pattern classification — as opposed to the use of a deterministic c_T.

We will use an extension to PAC learning in which, instead of considering T_k as having been generated using a distribution P on X and the target concept c_T, we consider T_k to be generated using only a distribution P' on $X \times \mathbb{B}$, from which examples are drawn independently. If a network computes the function $f_{\mathbf{w}}$ then we define the error of $f_{\mathbf{w}}$ with respect to P' as,

$$\begin{aligned} \mathrm{er}_{P'}(f_{\mathbf{w}}) &= \Pr\left[f_{\mathbf{w}}(\mathbf{x}) \neq o\right] \\ &= P'[\{(\mathbf{x}, o) \mid f_{\mathbf{w}}(\mathbf{x}) \neq o\}] \end{aligned} \tag{3.15}$$

where $(\mathbf{x}, o)$ is a random example. It may thus not be possible to define a deterministic $c_T \subseteq X$ as given $\mathbf{x} \in X$ both $(\mathbf{x}, +1)$ and $(\mathbf{x}, -1)$ may have non-zero probability. Using this extension we can, for example, model the situation in which examples are generated as in the standard PAC learning formalism but T_k is modified by some random process which alters $\mathbf{x}_i$ or o_i.

The work of [73] allows us to model the task of searching for a deterministic hypothesis $h_{\mathbf{w}}$ which, with high probability, is a good approximation to some stochastic target concept. This corresponds closely to the usual manner in which the problem of training a connectionist network is cast. In applying this extended version of PAC learning, we will use two important theorems due to Baum and Haussler [79] and based on the work of Vapnik [78], Blumer *et al.* [73] and Ehrenfeucht *et al.* [85]. These theorems are stated and applied in chapter 5 (theorems 5.2 and 5.4), and in common with standard PAC learning rely heavily on the growth function and VC dimension.

3.2.5 Further Results in Computational Learning Theory

Several extensions exist to the computational learning theory presented above, the most important of which we now briefly review. Many of the extensions mentioned here are reviewed in more detail by Anthony and Biggs [86].

The theory presented above clearly applies only to networks used to solve two-class pattern classification problems, which in general have only a single output. Extensions allowing us to consider, using PAC learning techniques, networks with many outputs, which can solve pattern classification problems with several classes, have been made by Shawe-Taylor and Anthony [87]. Extensions to the theory allowing us to deal with networks having real-valued outputs have been made by Haussler [88, 89].

One of the most desirable properties of the theory is that it provides results which are entirely independent of the actual probability distribution that governs the occurrence of training and testing examples. Unfortunately, this generality, while leading to results that are obviously very powerful, tends also to lead to results which are impractical, and we will pursue this point further in chapter 5. Extensions to the theory have been made which allow us to consider distribution-dependent learning, in which we deal with some particular class of probability distributions, or even a single, specific distribution, rather than with arbitrary distributions.

Extensions have also been made to PAC learning theory which allow a learning algorithm to make *queries*; for example, the algorithm may now be allowed to ask whether some particular input $\mathbf{x}$ is in class 1 or class 2 (Angluin [81]). Kearns and Schapire [90] have considered further methods for dealing with probabilistic concepts, and Fischer *et al.* [91] have attempted to incorporate Bayes decision theory into a PAC learning framework. Haussler *et al.* [92] have attempted to unite computational learning theory and the theories of generalization based on statistical physics which we introduce below. We discuss the work in [92] further in chapter 5.

3.3 Other Approaches to the Theoretical Analysis of Generalization

We now briefly summarize some alternative approaches to the theoretical analysis of generalization. As we mentioned at the beginning of this chapter, the most important alternative techniques are based on statistical physics. These techniques address the average generalization ability of a network whereas computational learning theory provides worst case results. We will not provide a detailed account of the former techniques; for a good introduction see Hertz *et al.* [15] and for further detail see Seung *et al.* [93] and references therein. The relationship between these methods and computational learning theory is also discussed in further detail in [93].

Kanaya and Miyake [94, 95] have proposed a definition of valid generalization based on Bayes decision theory, and have analysed the number of training examples required to produce valid generalization under this definition. Anshelevich *et al.* [96] have proposed a method for analysing generalization ability using very simple information theoretic arguments; although their theory contains many assumptions, which are not necessarily realistic, they find that it agrees well with simulation results for some simple problems. Further, quite simple analyses are given by Cover [17], Wan [97] and Denker *et al.* [98]. A rather different approach to generalization is given by Wolpert [99, 100], who introduces a model independent formalism involving the specification of various criteria required of a system that generalizes. For example, the system might be required to be invariant to particular types of coordinate transformation of the input space. A system is then constructed with these criteria in mind; systems constructed in this manner have been applied with encouraging results (Wolpert [40]). Finally, an analysis using the Vapnik-Chervonenkis and other inequalities appears in Devroye [101], and MacKay [102, 103] briefly discusses generalization in the context of his Bayesian framework.

3.4 The Importance of Capacity

An important point, which is supported by most theoretical work on generalization, is that we should in general attempt to use a network which has the minimum capacity necessary to provide acceptable performance on the training examples, where capacity is some appropriate measure of the 'size' of the class of functions computed by the network. This has been noted by various authors, such as Le Cun [104], and is reinforced by experimental studies (see for example Sietsma and Dow [105]). Simple algorithms for tuning the capacity of particular classifiers in this manner have been proposed by Boser *et al.* [11] and Guyon *et al.* [106]. A widely applicable technique for capacity tuning (although one that is quite computationally intensive) is that of *cross-validation* (see Efron [107]). (Note however that this technique will not necessarily select a network having strictly minimum capacity.)

There is a simple intuitive explanation for this fact: if we train a network with much larger capacity than necessary, then we can expect the probability that it performs well on the training data to be high. However, we can also expect that the *number* of different functions that the network can compute which perform well on the training examples will be high, and consequently that the probability of choosing the one which provides the 'best' generalization is low. Of course, we can increase the latter probability by increasing the number of training examples and hence reducing the number of functions that perform well thereon.

3.5 Conclusion

In this chapter we have introduced the theoretical framework that will be used in the next two chapters to investigate the ability of Φ-nets to generalize, and we have briefly reviewed the alternative formalisms that are available for the theoretical analysis of generalization. One reason that we have chosen to use a formalism based on PAC learning theory is that it allows us to derive results relating to the performance of a network in the worst case, rather than the average case. In order to apply the relevant theory, we need to deduce values for the growth functions and VC dimensions of the networks of interest, or bounds thereon; this is the main aim of the next chapter.

Chapter 4

The Generalization Ability of Φ-Nets I: Growth Functions and VC Dimensions

4.1 Introduction

In this and the following chapter we present a detailed study, using extended PAC learning theory, of the ability of Φ-nets to perform generalization. In the last chapter we saw how the growth function and the VC dimension of a network can be used in order to investigate its generalization ability. This chapter presents an investigation of the growth function and VC dimension of both general and specific Φ-nets, with both fixed and adapting basis functions. We also demonstrate that these parameters do not allow us to make a meaningful analysis of the generalization ability of connectionist networks trained using a self-structuring training algorithm, and we introduce a method which allows us to perform an appropriate analysis for Φ-nets with fixed basis functions in this case. We use our results in the next chapter to derive necessary and sufficient conditions on the number of training examples required by a Φ-net in order to obtain a particular generalization performance.

In section 4.2 we provide bounds on growth functions and VC dimensions for entirely general Φ-nets with fixed basis functions, and we introduce the idea of a restricted Φ-net, for which we provide further bounds. We then consider the case of networks trained using self-structuring. In section 4.3 we consider the specific case of radial basis function networks with fixed and adapting basis functions, for which we deduce results on the VC dimension for several commonly encountered types of basis function. In section 4.4 we use the fact that the VC dimension can be regarded as a measure of capacity to investigate the criticism that Φ-nets can in practice require a relatively large number of weights. Section 4.5 discusses some of our results and section 4.6 concludes the chapter.

Throughout this and the following chapter we assume that Φ-nets have $W = m + 1 \geq 2$ weights unless otherwise stated. Recall also that we deal only with networks which compute functions $f_{\mathbf{w}} : \mathbb{R}^n \to \mathbb{B}$. Some of the results presented in this and the following chapter have been published previously, or are accepted for publication (Holden and Rayner [108, 109], Holden and Anthony [110], Holden [111, 112], and Anthony and Holden [113, 114]) although the exposition provided here is considerably expanded.

4.2 Bounds on the Growth Function and VC Dimension of Φ-Nets

4.2.1 General Φ-Nets with Fixed Basis Functions

We begin by introducing the class $\mathcal{F}_n$ of *linear threshold functions* on $\mathbb{R}^n$.

Definition 4.1 *A* linear threshold function $f_{\mathbf{w}} : \mathbb{R}^n \to \mathbb{B}$ *is a function of the form* $f_{\mathbf{w}}(\mathbf{x}) = \rho[w_0 + w_1x_1 + \cdots + w_nx_n]$ *where* $w_i \in \mathbb{R}$ *for* $i = 0, 1, \ldots, n$*. The class* $\mathcal{F}_n$ *is defined as,*

$$\mathcal{F}_n = \{f_{\mathbf{w}} \mid \mathbf{w} \in \mathbb{R}^{n+1}\}. \tag{4.1}$$

The class $\mathcal{F}_n$ has a corresponding hypothesis space which can be identified with the set of all closed halfspaces on $\mathbb{R}^n$; clearly, it also corresponds to the usual class of linear discriminant functions with $\mathcal{C} = \mathbb{B}$. The following result has been proved by several different authors, although the best known derivation is probably that due to Wenocur and Dudley [115].

Lemma 4.2 *The class* $\mathcal{F}_n$ *of linear threshold functions on* $\mathbb{R}^n$ *has VC dimension* $\mathcal{V}(\mathcal{F}_n) = n+1$.

Similarly, it is known [71] that $\triangle_{\mathcal{F}_n}(i) = 2\Theta(n, i-1)$ for $i \geq 2$, where Θ is the function defined in Sauer's lemma (lemma 3.6).

Consider now a threshold Φ-net which computes a class of functions $\mathcal{F}_n^{\Phi}$. As we have previously noted, this network operates by mapping input vectors into a new space — the extended space — and computing a halfspace on this space. Intuitively therefore, we might expect the VC dimension $\mathcal{V}(\mathcal{F}_n^{\Phi})$ of the Φ-net to be bounded above with an upper bound equal to the VC dimension of the set of all halfspaces on the new space, or $\mathcal{V}(\mathcal{F}_m)$ where $\mathcal{F}_m$ is as defined above and, as usual, $m = \mid \Phi \mid$. For the same reason, we would also expect $\triangle_{\mathcal{F}_n^{\Phi}}(i)$ to have an upper bound of $\triangle_{\mathcal{F}_m}(i)$. This intuition is in fact true and is formalized in lemma 4.4 below. In order to prove this we use the following result.

Lemma 4.3 *For any hypothesis space* H*,* $\triangle_H(i) \leq \triangle_H(i+1)$.

Proof Consider some set of points S_i and the corresponding set $\triangle_H(S_i)$. If a new random point is added to S_i to give S_{i+1} then $\triangle_H(S_{i+1})$ must contain at least one dichotomy of S_{i+1} for every dichotomy of S_i induced by some $h \in H$, as well as any new dichotomies induced. Thus, $\mid \triangle_H(S_i) \mid \leq \mid \triangle_H(S_{i+1}) \mid$ which implies that $\triangle_H(i) \leq \triangle_H(i+1)$. □

We can now prove the following, which formalizes the intuition suggested above. An alternative way of proving the first part of this lemma, based on an existing result, will be given below. We include the following version of the proof as it is new, and as we feel that it provides a better insight into the problem of interest.

Lemma 4.4 (Upper bounding lemma) *Given any* Φ*-net, which computes the class of functions* $\mathcal{F}_n^{\Phi}$*, it is true that* $\mathcal{V}(\mathcal{F}_n^{\Phi}) \leq \mathcal{V}(\mathcal{F}_m) = W$ *and that* $\triangle_{\mathcal{F}_n^{\Phi}}(i) \leq \triangle_{\mathcal{F}_m}(i) = 2\Theta(m, i-1)$ *for* $i \geq 2$ *regardless of the value of* m *and the actual basis functions used.*

Proof

The VC dimension bound It is clearly possible to have $\mathcal{V}(\mathcal{F}_n^\Phi) = \mathcal{V}(\mathcal{F}_m)$. This is true, for example, if the extended vector is formed as $\tilde{\mathbf{x}}^T = [\ 1 \quad \mathbf{x}^T\ ]$. The case $\mathcal{V}(\mathcal{F}_n^\Phi) < \mathcal{V}(\mathcal{F}_m)$ is possible if the set of basis functions chosen is suitably limiting, for example, if we use $\tilde{\mathbf{x}}^T = [\ 1 \quad 1 \quad \cdots \quad 1\ ]$. The case $\mathcal{V}(\mathcal{F}_n^\Phi) > \mathcal{V}(\mathcal{F}_m)$ is not possible; this is a direct consequence of the definition of the VC dimension.

The growth function bound Application of the above methods of constructing $\tilde{\mathbf{x}}$ establishes that $\triangle_{\mathcal{F}_n^\Phi}(i) \leq \triangle_{\mathcal{F}_m}(i)$. A careful consideration of the case $\triangle_{\mathcal{F}_n^\Phi}(i) > \triangle_{\mathcal{F}_m}(i)$ shows that this would only be possible if the mapping φ (as defined in chapter 2) maps two or more points in the input space into a single point in the space spanned by the basis functions *and*,

$$\triangle_{\mathcal{F}_m}(j-k) > \triangle_{\mathcal{F}_m}(j), \tag{4.2}$$

for some positive integers j and k where $k < j$. The inequality of equation 4.2 is not possible by lemma 4.3 and so the growth function bound is proved.

□

The upper bound provided by lemma 4.4 is clearly met for a threshold Φ-net which computes linear discriminant functions, but is it met by more general threshold Φ-nets? We can partly provide an answer to this question using the following result which is originally due to Dudley [116], and appears in the following form in [110].

Theorem 4.5 *Let $\mathcal{S}$ be a vector space*[1] *of real-valued functions defined on some set X, and suppose that the dimension of the space $\mathcal{S}$ is d. For functions $s \in \mathcal{S}$ define the function $s^+ : X \to \mathbb{B}$ as $s^+ = \rho \circ s$, and define $\mathcal{S}^+ = \{s^+ \mid s \in \mathcal{S}\}$. Then the VC dimension $\mathcal{V}(\mathcal{S}^+)$ of $\mathcal{S}^+$ is d.*

This theorem tells us several things. Firstly, for a given set Φ of basis functions, it is easily verified that the class of real-valued functions,

$$\mathcal{F} = \left\{ f_{\mathbf{w}}(\mathbf{x}) = w_0 + \sum_{i=1}^{m} w_i \phi_i(\mathbf{x}) \mid \phi_i \in \Phi, \mathbf{x} \in \mathbb{R}^n \text{ and } w_i \in \mathbb{R} \text{ for } i = 0, 1, \ldots, m \right\} \tag{4.3}$$

is a vector space of functions on $\mathbb{R}^n$ which has dimension $m+1$ if the set of functions $\{1, \phi_1, \ldots, \phi_m\}$ is linearly independent and has dimension less than $m+1$ otherwise. Note that for this set of functions to be linearly independent we require that there are no constants $c_i \in \mathbb{R}$, $i = 0, 1, \ldots, m$, where at least one c_i is non-zero, such that,

$$c_0 + c_1\phi_1(\mathbf{x}) + \cdots + c_m\phi_m(\mathbf{x}) = 0 \tag{4.4}$$

for all $\mathbf{x} \in \mathbb{R}^n$. Clearly therefore the upper bound of lemma 4.4 is met for any threshold Φ-net for which the set of functions $\{1, \phi_1, \ldots, \phi_m\}$ is linearly independent; this fact is used in [110] to show that threshold Φ-nets computing polynomial discriminant functions meet the upper bound (see also Young and Downs [118]). It can also be used to provide an alternative proof of the VC dimension bound of lemma 4.4, and to prove lemma 4.2.

Unfortunately, proving that a given set of functions is linearly independent is usually by no means a trivial problem, although it is obviously sensible to use such a set of functions if possible, as otherwise some of the functions are effectively redundant.

[1]For completeness, we re-state the full definition of a vector space (Green [117]). A *vector space* is a structure consisting of two sets and their associated operations such that one, the elements of which are called *vectors*, forms an Abelian group, and the other, the elements of which are called *scalars*, forms a field. Furthermore, the operation of *scalar multiplication* is defined which produces a vector as the product of a scalar and a vector, distributes over addition of both vectors and scalars, and is associative with multiplication of scalars.

4.2.2 Restricted Φ-Nets with Fixed Basis Functions

In the next chapter, we will need lower bounds as well as upper bounds on the VC dimension of a given Φ-net. We saw above that for Φ-nets in which the relevant functions are linearly independent the VC dimension is equal to the number of weights W in the network. If we cannot prove that this is the case then finding a lower bound may be difficult; it is not possible for the completely general class of Φ-nets to derive a good lower bound on $\mathcal{V}(\mathcal{F}_n^\Phi)$ although a trivial bound of $\mathcal{V}(\mathcal{F}_n^\Phi) \geq 1$ is obtained easily, again by considering the use of a network which can only compute the extended vector $\tilde{\mathbf{x}}^T = [\ 1 \quad 1 \quad \cdots \quad 1\ ]$. A simple way to obtain lower bounds is to consider classes of Φ-nets in which the set of basis functions has been restricted in some manner. Many of the standard Φ-nets introduced in chapter 2 contain basis functions which correspond to terms in a polynomial in addition to some further basis functions, and we therefore introduce the following definition.

Definition 4.6 (Restricted Φ-net) *A* restricted Φ-net *is a standard Φ-net in which n of the basis functions have the form $\phi_i(\mathbf{x}) = x_i$. In this case the extended vector is of the form,*

$$\tilde{\mathbf{x}}^T = [\ 1 \quad \mathbf{x}^T \quad \phi_{n+1}(\mathbf{x}) \quad \cdots \quad \phi_m(\mathbf{x})\ ]. \tag{4.5}$$

LDFs and PDFs are clearly restricted Φ-nets, as are some radial basis functions networks and regularization networks. We can now derive a lower bound on $\mathcal{V}(\mathcal{F}_n^\Phi)$ as follows.

Lemma 4.7 *The VC dimension of a restricted Φ-net has a lower bound of $n+1$.*

Proof If we set $w_i = 0$ for $i = n+1, \ldots, m$, that is, we set the weights corresponding to the basis functions $\phi_{n+1}(\mathbf{x})$ to $\phi_m(\mathbf{x})$ in equation 4.5 to zero, the network computes the class $\mathcal{F}_n$ of halfspaces in the input space when we vary the remaining weights. Thus $\mathcal{V}(\mathcal{F}_n) = n+1$ provides a lower bound for $\mathcal{V}(\mathcal{F}_n^\Phi)$. □

Let $\mathcal{F}_{PDF}(n,d)$ be the class of polynomial threshold functions on $\mathbb{R}^n$ computed by an (n,d) discriminator. In [110] it is proved that for this type of classifier,

$$\mathcal{V}(\mathcal{F}_{PDF}(n,d)) = \left(\begin{array}{c} n+d \\ d \end{array} \right). \tag{4.6}$$

Clearly we can use this result to provide a lower bound of,

$$\mathcal{V}(\mathcal{F}_n^\Phi) \geq \left(\begin{array}{c} n+d \\ d \end{array} \right) \tag{4.7}$$

on the VC dimension of more general types of restricted Φ-net in which a subset of the basis functions corresponds to those used to compute the polynomial threshold functions. In an even more general sense, if we know the exact VC dimension of a Φ-net constructed using a particular set Φ_1 of basis functions, then we can immediately obtain a lower bound on the VC dimension of any Φ-net constructed using a set $\Phi = \{\Phi_1, \Phi_2\}$ of basis functions where Φ_2 is any further suitable set.

4.2.3 Φ-Nets with Self-Structuring

In the last chapter we noted that the capacity of a network is directly related to its generalization performance. We therefore expect self-structuring to be advantageous in constructing networks which generalize well, as the capacity of a network is in general directly related to its structure and we can therefore design self-structuring training algorithms which attempt to optimize capacity. In general these algorithms will attempt to minimize capacity and consequently to minimize the size of a network (where 'size' is appropriately defined).

Self-structuring can be approached in several different ways, which we will summarize in chapter 6. In this dissertation we are concerned with one particular approach, in which we begin with a 'large' network and then attempt to remove weights which are not required; again, the reasons for adopting this approach are discussed in chapter 6. In other words, given a Φ-net which computes the class of functions $\mathcal{F}_n^\Phi$, our self-structuring algorithms bias the search for an optimum weight vector in favour of a vector with many zero elements. Now, it is important to note that such algorithms search over precisely the same class of functions, namely $\mathcal{F}_n^\Phi$, as standard training algorithms which make no attempt to adapt the structure of the network. It is therefore not possible to directly analyse the effect of this type of self-structuring on generalization performance within this particular formalism, because bounds on the growth function and VC dimension for $\mathcal{F}_n^\Phi$ are not altered in any way as a result of its use. MacKay [103] has made effectively the same criticism against the use of the VC dimension. We can however gain some insight into the effect of this type of self-structuring by restricting $\mathcal{F}_n^\Phi$ in a manner which takes into account and sensibly reflects the bias in favour of weight vectors with many zero elements. In order to facilitate an analysis we introduce the concept of an *l-restriction*, which was inspired by a similar technique used in [79] (see section 5.4). This concept will be used in our analysis in the next chapter.

Definition 4.8 (l-restriction) *Given a class of functions $\mathcal{F}_n^\Phi$ computed by some Φ-net having $W \geq 3$ weights, we define its l-*restriction *$\mathcal{R}^l(\mathcal{F}_n^\Phi) \subseteq \mathcal{F}_n^\Phi$ where $l \geq 2$ and $W > l$ as the subset of $\mathcal{F}_n^\Phi$ containing the functions computed by the Φ-net when l weights are unconstrained, one of which corresponds to the unit term in the extended vector, and the remaining weights are fixed equal to zero. Formally, if the weight vector of the Φ-net is $\mathbf{w}^T = [\ w_0\ \ w_1\ \ \cdots\ \ w_m\ ]$, let $\mathcal{W}^{(l-1)}$ be the set of all subsets of weights which do not contain w_0 and which have cardinality $(l-1)$. For example, if $m = 10$ a possible element of $\mathcal{W}^3$ is $\{w_2, w_5, w_{10}\}$. Then $\mathcal{R}^l(\mathcal{F}_n^\Phi)$ is the class of functions,*

$$\mathcal{R}^l(\mathcal{F}_n^\Phi) = \left\{ f_{\mathbf{w}}(\mathbf{x}) = \rho \left[w_0 + \sum_{w_i \in \omega} w_i \phi_i(\mathbf{x}) \right] \mid \omega \in \mathcal{W}^{(l-1)} \text{ and } w_i \in \mathbb{R} \text{ for } i = 0, 1, \ldots, m \right\}. \tag{4.8}$$

Intuitively, the l-restriction of a Φ-net is simply the set of all functions implemented by the Φ-net when it is forced to have *at most* l non-zero weights. The requirement that w_0 is always unconstrained is needed in the proof of theorem 5.8 in order to ensure that the network still computes general closed halfspaces on the extended space. In this theorem we will also need to know the number $N(l, W)$ of ways of choosing a set of unconstrained weights. As w_0 is always unconstrained $N(l, W)$ is simply the number of ways of choosing $(l-1)$ weights from $(W-1)$, without distinguishing according to the order of choice or allowing the same weight to be chosen more than once, or (see Biggs [119]),

$$N(l, W) = \binom{W-1}{l-1}. \tag{4.9}$$

Note that the l-restriction of a Φ-net can be written as the union of the classes of functions computed by $N(l, W)$ smaller Φ-nets:

$$\mathcal{R}^l(\mathcal{F}_n^\Phi) = \bigcup_{i=1}^{N(l,W)} \mathcal{F}_n^{\Phi_i} = \mathcal{F}_n^{\Phi_1} \cup \mathcal{F}_n^{\Phi_2} \cup \cdots \cup \mathcal{F}_n^{\Phi_{N(l,W)}}. \tag{4.10}$$

In equation 4.10 each term $\mathcal{F}_n^{\Phi_i}$ corresponds to one possible way of choosing the non-zero weights, so we have $\Phi_i \subset \Phi$ where Φ_i is the subset containing the basis functions which do not correspond to zero weights and $\mid \Phi_i \mid = l - 1$. This will be important in theorem 5.8 and in appendix A.

Although we only concern ourselves with one type of self-structuring in this dissertation, it is relevant at this point to ask whether techniques involving the use of the growth function and VC dimension can be used effectively in circumstances where other forms of self-structuring are employed. The obvious alternative approach to self-structuring to that mentioned above is to begin with a small network and to increase its size during training; the next logical step is to allow network size to increase *and* decrease during training. Now, note that a method such as that introduced above can also sometimes be used to *approximately* model these cases, because we may still be searching for a 'small' network constructed using some available collection of resources (in the case of Φ-nets the collection of resources is the set Φ of basis functions). This fact has not previously been noted.

However, what happens if the collection of available resources is not fixed in advance, or is infinite (for example, if we allow a self-structuring algorithm to add a radial basis function to a network *and* specify its centre)? Furthermore, how can we *exactly*, rather than approximately, model self-structuring? Ultimately, the situation can become very complicated, because the class of functions which we must deal with can *change* during training in a manner dependent on the specific training data available. Research to date in computational learning theory has not attempted to fully model these situations[2], which are clearly important, and we therefore propose this as an important subject for further research.

4.3 A Specific Case: Radial Basis Function Networks

We demonstrated above that the VC dimension of a completely general Φ-net must be such that $\mathcal{V}(\mathcal{F}_n^{\Phi}) \leq W$; we also demonstrated that $\mathcal{V}(\mathcal{F}_n^{\Phi}) = W$ if $\{1, \phi_1, \phi_2, \ldots, \phi_m\}$ is a linearly independent set of functions. We now investigate the VC dimension of various types of radial basis function network. We mentioned briefly in chapter 2 that these networks have an excellent mathematical foundation in interpolation theory, and it is by exploiting this fundamental work that we obtain our results. We therefore begin with a brief review of the theory required; our review is based on a review given in [44], however we have added some further material and made some corrections.

Recall that a general RBFN computes a class of functions of the form $f_{\mathbf{w}} = \rho \circ \overline{f_{\mathbf{w}}}$ where $\overline{f_{\mathbf{w}}} : \mathbb{R}^n \rightarrow \mathbb{R}$ is of the form,

$$\overline{f_{\mathbf{w}}}(\mathbf{x}) = \sum_{i=1}^{p} \lambda_i \phi(\|\mathbf{x} - \mathbf{y}_i\|) + \sum_{i=1}^{q} \theta_i \psi_i(\mathbf{x}) \text{ for } q \leq p. \tag{4.11}$$

We will call functions of the form of $f_{\mathbf{w}} = \rho \circ \overline{f_{\mathbf{w}}}$ *radial threshold functions*. In equation 4.11, $\mathbf{w}^T = [\ \lambda_1 \ \ \lambda_2 \ \ \cdots \ \ \lambda_p \ \ \theta_1 \ \ \theta_2 \ \ \cdots \ \ \theta_q\ ]$ is a vector of weights, $\phi : \mathbb{R}_0^+ \rightarrow \mathbb{R}$ is a basis function, the vectors $\mathbf{y}_i \in \mathbb{R}^n$ are the centres of the basis functions, $\|.\|$ is the Euclidean norm and $\{\psi_i \mid i = 1, \ldots, q\}$ is a basis of the linear space of algebraic polynomials from $\mathbb{R}^n$ to $\mathbb{R}$ of degree at most $(d-1)$, which we denote by $\pi_{d-1}(\mathbb{R}^n)$, for some given d. In practice the polynomial terms are often not included and in this case the functions $\overline{f_{\mathbf{w}}}$ are of the form,

$$\overline{f_{\mathbf{w}}}(\mathbf{x}) = \sum_{i=1}^{p} \lambda_i \phi(\|\mathbf{x} - \mathbf{y}_i\|). \tag{4.12}$$

[2]The idea of an *Occam Algorithm* [71] provides a starting point, but does not capture the full potential complexity involved.

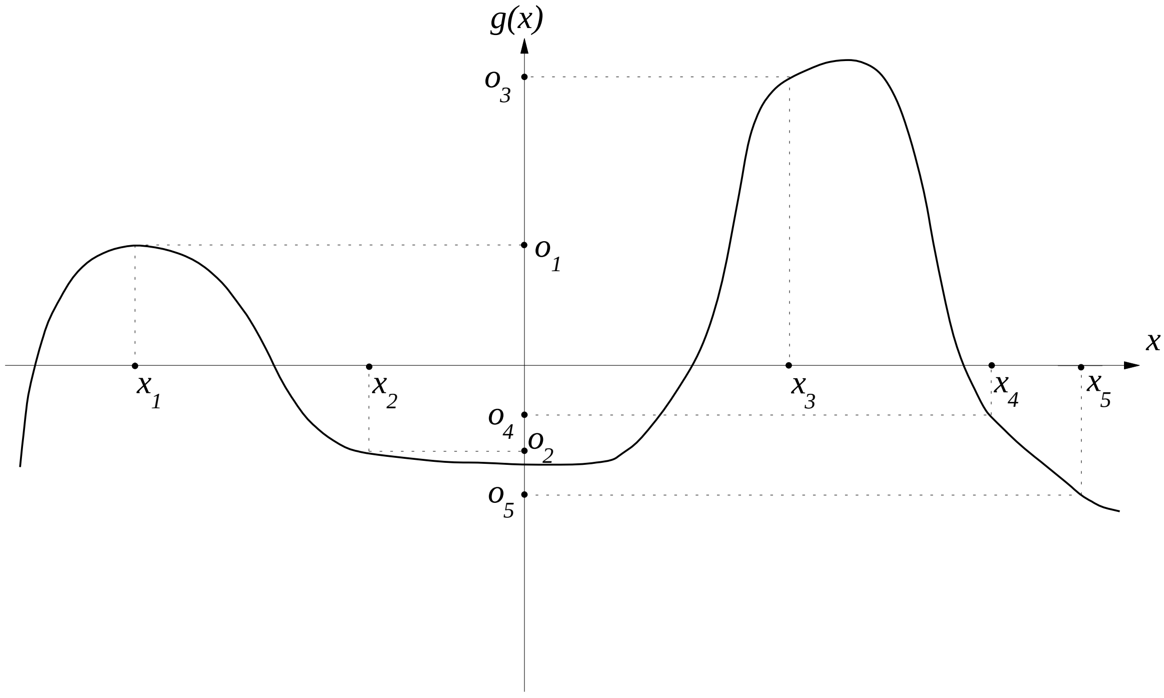


Figure 4.1: Forming a dichotomy of a set $S = \{x_1, x_2, x_3, x_4, x_5\}$ of points in $\mathbb{R}$ by interpolation using a real-valued function $g(x)$. The dichotomy formed by $\rho \circ g$ in this case is $(\{x_1, x_3\}, \{x_2, x_4, x_5\})$. Provided there is some $g \in \mathcal{G}$ which performs the interpolation regardless of the values of the o_i, $\mathcal{G}' = \{\rho \circ g \mid g \in \mathcal{G}\}$ shatters S.

A constant offset term λ_0 is often added in this case, but is omitted here.

4.3.1 Interpolation, Micchelli's Theorems and the VC Dimension

Consider now the problem of interpolating a given set $S_k = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$ of distinct points $\mathbf{x}_i \in \mathbb{R}^n$ with associated values $(o_1, o_2, \ldots, o_k)$ where the values $o_i \in \mathbb{R}$ can be chosen *arbitrarily*. That is, we wish to find a function g such that,

$$g(\mathbf{x}_i) = o_i \text{ for } i = 1, 2, \ldots, k. \tag{4.13}$$

Let us assume that g is chosen from a class of functions $\mathcal{G}$, and define the associated class of functions $\mathcal{G}'$ as $\mathcal{G}' = \{\rho \circ g \mid g \in \mathcal{G}\}$. Now, assume that we have a *particular* set of k points S_k to which we can assign arbitrary quantities $o_i \in \mathbb{R}$. Clearly, if it is possible to prove that given S_k there exists a $g \in \mathcal{G}$ which performs the interpolation *regardless* of the actual values o_i used then $\mathcal{V}(\mathcal{G}') \geq k$. To see why this is so consider any dichotomy (S_k^+, S_k^-) of the set S_k. We can pick the o_i such that when $\mathbf{x}_i \in S_k^+$ the corresponding o_i is an arbitrary positive quantity and when $\mathbf{x}_i \in S_k^-$ the corresponding o_i is an arbitrary negative quantity. Because there must exist some $g \in \mathcal{G}$ which performs the corresponding interpolation, $g' = \rho \circ g$ must induce the dichotomy, and because we can start with *any* dichotomy, $\mathcal{G}'$ shatters S_k. This is illustrated in figure 4.1.

The reason that functions of the form of $\overline{f_{\mathbf{w}}}$ as defined in equation 4.11 are useful is that it is always possible to interpolate points in such a set S_k for arbitrary o_i using this type of function if $p = k$ and the centres $\mathbf{y}_i$ correspond to the points $\mathbf{x}_i$, that is,

$$\overline{f_{\mathbf{w}}}(\mathbf{x}) = \sum_{i=1}^{k} \lambda_i \phi(\|\mathbf{x} - \mathbf{x}_i\|) + \sum_{i=1}^{q} \theta_i \psi_i(\mathbf{x}) \text{ where } q \leq k, \tag{4.14}$$

provided the basis function ϕ satisfies a condition which we discuss below. We can now define the above class of functions $\mathcal{G}$ as,

$$\mathcal{G} = \{\overline{f_{\mathbf{w}}} \mid \mathbf{w} \in \mathbb{R}^{k+q}\}. \tag{4.15}$$

As in chapter 2, we note that under the constraints of equation 4.13 we now have a set of k linear equations with $(k + q)$ coefficients; we fix the remaining degrees of freedom by requiring that,

$$\sum_{i=1}^{k} \lambda_i \psi_j(\mathbf{x}_i) = 0 \text{ where } j = 1, \ldots, q. \tag{4.16}$$

A sufficient condition on the basis function ϕ for the existence of a suitable interpolating function is that $\phi \in \mathbb{P}_d(\mathbb{R}^n)$ where we define $\mathbb{P}_d(\mathbb{R}^n)$ to be the set of *strictly conditionally positive definite (SCPD) functions of order d* (Micchelli [43]).

Definition 4.9 *Let h be a continuous function on $[0, \infty)$. The function h is SCPD of order d on $\mathbb{R}^n$ if for any set S_k of k distinct points $\mathbf{x}_i \in \mathbb{R}^n$ and scalars $c_1, c_2, \ldots, c_k \in \mathbb{R}$ (at least one $c_i \neq 0$) where,*

$$\sum_{i=1}^{k} c_i \psi(\mathbf{x}_i) = 0 \tag{4.17}$$

for all $\psi \in \pi_{d-1}(\mathbb{R}^n)$ the quadratic form $\sum_{i=1}^{k} \sum_{j=1}^{k} c_i c_j h(\|\mathbf{x}_i - \mathbf{x}_j\|)$ is positive. In the special case of $d = 0$ the class of SCPD functions is the class of functions for which $\sum_{i=1}^{k} \sum_{j=1}^{k} c_i c_j h(\|\mathbf{x}_i - \mathbf{x}_j\|)$ is positive.

We now define $\mathbb{P}_d$ to be the set of functions which are in $\mathbb{P}_d(\mathbb{R}^n)$ over any $\mathbb{R}^n$,

$$\mathbb{P}_d = \bigcap_{n \geq 1} \mathbb{P}_d(\mathbb{R}^n). \tag{4.18}$$

A theorem due to Micchelli provides us with a simple means of determining whether a particular basis function ϕ is in $\mathbb{P}_d$, and consequently whether it is a good basis function to use in forming $\overline{f_{\mathbf{w}}}$. Before stating the theorem we need to introduce the idea of a *completely monotonic* function.

Definition 4.10 *A function h is completely monotonic on $(0, \infty)$ if it is in $C^\infty(0, \infty)$ and its sequence of derivatives is such that,*

$$(-1)^i h^{(i)}(x) \geq 0 \tag{4.19}$$

for $x \in (0, \infty)$ and $i = 0, 1, 2, \ldots$.

Theorem 4.11 (Micchelli [43], Dyn and Micchelli [120]) *If a function $h(r)$ is continuous on $[0, \infty)$, $h(r^2) \in C^\infty(0, \infty) \cap C[0, \infty)$ and $(-1)^d h^{(d)}$ is completely monotonic on $(0, \infty)$ but not constant then $h(r^2) \in \mathbb{P}_d$.*

Consider now the special case mentioned above, which corresponds more closely to the types of network used in practice, in which we form $\overline{f_{\mathbf{w}}}$ as,

$$\overline{f_{\mathbf{w}}}(\mathbf{x}) = \sum_{i=1}^{p} \lambda_i \phi(\|\mathbf{x} - \mathbf{y}_i\|). \tag{4.20}$$

Once again, we will insist that $p = k$ basis functions are used and,

$$\mathbf{y}_i = \mathbf{x}_i \text{ for } i = 1, \ldots, k. \tag{4.21}$$

Clearly in this case the interpolation is possible if there exists a solution to the set of equations,

$$\mathbf{o} = \begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ o_k \end{bmatrix} = \begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1k} \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{k1} & \phi_{k2} & \cdots & \phi_{kk} \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_k \end{bmatrix} = \boldsymbol{\phi}\boldsymbol{\lambda} \tag{4.22}$$

where $\phi_{ij} = \phi(\|\mathbf{x}_i - \mathbf{x}_j\|)$. Thus we would like $\boldsymbol{\phi}$ to be non-singular so that $\boldsymbol{\lambda} = \boldsymbol{\phi}^{-1}\mathbf{o}$. It is possible to show that $\boldsymbol{\phi}$ is non-singular if ϕ is SCPD of order 0 [42, 44]. In some cases theorem 4.11 can therefore be used to determine whether a given ϕ can be used successfully. Alternatively we can use another theorem due to Micchelli.

Theorem 4.12 (Micchelli [43]) *If a function h is continuous on $[0, \infty)$, positive on $(0, \infty)$, and has a first derivative that is completely monotonic but not constant on $(0, \infty)$, then for any set S_k of k vectors $\mathbf{x}_i \in \mathbb{R}^n$ where n is arbitrary,*

$$(-1)^{k-1} \det h(\|\mathbf{x}_i - \mathbf{x}_j\|^2) > 0. \tag{4.23}$$

Now, if we choose a basis function ϕ such that $\phi(\sqrt{r})$ satisfies the conditions provided by theorem 4.12 it is not possible that $\det(\boldsymbol{\phi}) = 0$. This implies that $\boldsymbol{\phi}$ is non-singular as required.

4.3.2 RBFNs with Fixed Basis Functions

To briefly summarize the theory discussed above, consider again our original RBFNs as defined in equations 4.11 and 4.12. Provided we choose a suitable basis function ϕ using the conditions given in the appropriate theorem 4.11 or 4.12, an RBFN having distinct, *fixed* centres $\{\mathbf{y}_i\}$ where $i = 1, 2, \ldots, p$ shatters the set of p vectors $\{\mathbf{x}_i\}$ where $\mathbf{x}_i = \mathbf{y}_i$ for $i = 1, 2, \ldots, p$.

We now immediately obtain two corollaries, in which the various basis functions used are as defined in chapter 2; note that for basis functions which include a parameter c or σ, this parameter is fixed — it is not adapted during training. We have stated results only for the basis functions given in chapter 2, which tend to be the most commonly encountered in practice; the general theory given above can easily be used to provide similar results for other basis functions by applying precisely the same proof techniques that we employ below.

First, we consider simple RBFNs of the form $f_{\mathbf{w}}(\mathbf{x}) = \rho[\sum_{i=1}^{p} \lambda_i \phi(\|\mathbf{x} - \mathbf{y}_i\|)]$. As these networks are Φ-nets without a unit term in the extended vector we need the following result.

Lemma 4.13 *Consider the class $\hat{\mathcal{F}}_n^{\Phi}$ of functions $\hat{f}_{\mathbf{w}} : \mathbb{R}^n \to \mathbb{B}$ computed by a network where,*

$$\hat{f}_{\mathbf{w}}(\mathbf{x}) = \rho[\lambda_1 \phi_1(\mathbf{x}) + \cdots + \lambda_p \phi_p(\mathbf{x})]. \tag{4.24}$$

In equation 4.24, the λ_i are reals and the ϕ_i are fixed basis functions, and we therefore have a network identical to a standard Φ-net with the exception that it has no unit term in the extended vector. For this network, the VC dimension $\mathcal{V}(\hat{\mathcal{F}}_n^{\Phi})$ is such that $\mathcal{V}(\hat{\mathcal{F}}_n^{\Phi}) \leq p$.

Proof As the space of real-valued functions of the form $\sum_{i=1}^{p} \lambda_i \phi_i(\mathbf{x})$ can easily be shown to be a vector space of dimension at most p, this is a direct consequence of theorem 4.5. $\square$

Corollary 4.14 *Consider the radial basis function networks of the form,*

$$f_{\mathbf{w}}(\mathbf{x}) = \rho\left[\sum_{i=1}^{p} \lambda_i \phi(\|\mathbf{x} - \mathbf{y}_i\|)\right] \tag{4.25}$$

where the centres $\{\mathbf{y}_i\}$ are fixed and distinct and ϕ is one of the functions ϕ_{LIN}, ϕ_{GAUSS}, ϕ^*_{MQ} or ϕ^*_{IMQ} defined in chapter 2. In each case we have $\mathcal{V}(\hat{\mathcal{F}}_n^{\Phi}) = p$.

Proof By theorem 4.11, $\phi_{GAUSS} \in \mathbb{P}_0$ and $\phi^*_{IMQ} \in \mathbb{P}_0$. Also the functions $\sqrt{r}$ and $(r + c^2)^{\beta}$ where $0 < \beta < 1$ satisfy the conditions provided by theorem 4.12 [44]. Thus in all four cases we have $\mathcal{V}(\hat{\mathcal{F}}_n^{\Phi}) \geq p$ by the arguments given above. Also, we demonstrated in lemma 4.13 that for this type of network $\mathcal{V}(\hat{\mathcal{F}}_n^{\Phi}) \leq p$ and thus we must have $\mathcal{V}(\hat{\mathcal{F}}_n^{\Phi}) = p$. $\square$

We now consider the more general RBFNs as defined earlier.

Corollary 4.15 *Consider the general radial basis function networks of the form,*

$$f_{\mathbf{w}}(\mathbf{x}) = \rho\left[\sum_{i=1}^{p} \lambda_i \phi(\|\mathbf{x} - \mathbf{y}_i\|) + \psi(\boldsymbol{\theta}, \mathbf{x})\right] \tag{4.26}$$

where $\psi(\boldsymbol{\theta}, \mathbf{x})$ is a degree 1 polynomial,

$$\psi(\boldsymbol{\theta}, \mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n, \tag{4.27}$$

the centres $\{\mathbf{y}_i\}$ are again fixed and distinct, $p \geq n + 1$, and ϕ is one of the functions ϕ_{CUB} or ϕ_{TPS}. The VC dimension of the network obeys $p \leq \mathcal{V}(\mathcal{F}_n^{\Phi}) \leq p + n + 1$.

Proof By theorem 4.11, $\phi_{CUB} \in \mathbb{P}_2$ and $\phi_{TPS} \in \mathbb{P}_2$ and so by the arguments given above we have $\mathcal{V}(\mathcal{F}_n^{\Phi}) \geq p$. The upper bound is a direct consequence of lemma 4.4 and the result follows. $\square$

4.3.3 RBFNs with Adapting Basis Functions

Although we are primarily interested in Φ-nets having fixed basis functions, we now briefly consider some Φ-nets having adapting basis functions. Specifically, we now consider the consequences of allowing the centres $\{\mathbf{y}_i\}$ of the basis functions in a radial basis function network to adapt during training; consequently, the number of weights is now $W_1 = p(n + 1)$ for the networks of equation 4.25 and $W_2 = p(n + 1) + n + 1$ for the networks of equation 4.26. First, we note that obviously the results given above immediately provide lower bounds of $\mathcal{V}(\mathcal{F}) \geq p$ on the VC dimension of networks of this type when appropriate basis functions ϕ are used. We also obtain the following simple result.

Corollary 4.16 *Consider the radial basis function networks of the types used in corollaries 4.14 and 4.15. If the centres $\{\mathbf{y}_i\}$ are allowed to adapt during training then the networks are all able to shatter any set of p distinct points. (Note that parameters c or σ in the basis functions are still fixed.)*

The proof of corollary 4.16 is simple — the networks can shatter the set of p points corresponding to the centres $\{\mathbf{y}_i\}$, and these centres can be placed anywhere. It is interesting that the p points are not required to be in any kind of *general position*[3] as is usually the case in similar results for other types of network (see for example Cover [17]). As the corollary shows that in this case the networks can shatter *any* set of p distinct points, we suggest that the lower bounds given above for these networks are not tight; it remains an open question whether they can be improved, for example it is interesting to ask whether it is possible to obtain lower bounds comparable to those of $\Omega(W \log_2 W)$ proved by Maass [121, 122] for certain types of feedforward network, which we discuss in detail in the next section.

We have been unable to obtain upper bounds on the VC dimension of radial basis function networks with adapting centres for all but the simplest cases. We note that if the basis function ϕ is of the form,

$$\phi(r) = r^i \tag{4.28}$$

where i is an even, positive integer then it is trivial to show that the network computes some subset of the set of functions computed by a Φ-net having *fixed* basis functions which are of the same form as those in a PDF. For example, if $i = 2$, $n = 2$ and $m = 1$ we could have,

$$\begin{aligned} f_{\mathbf{w}}(\mathbf{x}) &= \rho[\lambda_1 \phi(\|\mathbf{x} - \mathbf{y}_1\|)] \\ &= \rho[\lambda_1((x_1 - y_1)^2 + (x_2 - y_2)^2)] \\ &= \rho[\lambda_1(x_1^2 - 2x_1y_1 + y_1^2 + x_2^2 - 2x_2y_2 + y_2^2)] \\ &= \rho[(\lambda_1 y_1^2 + \lambda_1 y_2^2) - 2y_1\lambda_1 x_1 - 2y_2\lambda_1 x_2 + \lambda_1 x_1^2 + \lambda_1 x_2^2] \end{aligned} \tag{4.29}$$

and an upper bound on the VC dimension is easily obtained using lemma 4.4; in this case we have $\mathcal{V}(\mathcal{F}) \leq 5$.

A further question which provides a subject for further research is that of the effect of allowing σ to adapt when using the basis function ϕ_{GAUSS}. An answer to this question would be of significant importance on the basis of practical experience.

4.4 The VC Dimension as a Measure of Capacity

The VC dimension can be regarded as a measure of the capacity or 'expressive power' of a network; the larger the VC dimension of a network the more powerful we expect it to be and the more likely that it is capable of learning a given set of examples. On the other hand, as discussed in chapter 3, the number of examples which must be learnt before we can expect to obtain valid generalization can also be expected to increase with increasing capacity.

In this section we address the question of how the VC dimension of a Φ-net is likely to compare to that of an alternative type of network having the same number of weights. We ask this question for the following reason. As we mentioned in chapter 2, Φ-nets have been criticized for potentially requiring a relatively large number of weights, and we are interested in trying to determine whether this criticism will be valid in general, as it has not to date been the subject of practical or theoretical study. This criticism is most commonly applied to polynomial discriminant functions (recall that an (n, d) discriminator requires C_n^{n+d} weights); note however that it has also been found in practical comparisons with multilayer perceptrons that radial basis function networks with fixed centres can require relatively large numbers of weights (see [6] and references therein).

[3] A set S_k of k points in $\mathbb{R}^n$ is in *general position* if there is no subset of S_k containing $(j+1)$ points which lies on a $(j-1)$-dimensional hyperplane for $j = 1, 2, \ldots, n$.

We consider, as usual, Φ-nets with fixed basis functions, and compare them with feedforward networks of linear threshold elements. The latter networks are a type of multilayer perceptron, in which the nodes compute a weighted sum of their inputs and pass the result through a threshold operation rather than the more usual sigmoidal function. Note however that we discuss the use of sigmoids further below. We define these networks in a precise manner as follows (see also [79]).

Definition 4.17 (Feedforward Network) *A* Feedforward Network *is a directed acyclic graph G having an ordered sequence of n real-valued source nodes, called* inputs, *and a single sink node, called the* output. *We call any node which is not an input a* computation node, *and any computation node other than the output a* hidden node. *Each computation node n_i can compute a class of functions $\mathcal{F}_{n_i}$ where $f \in \mathcal{F}_{n_i}$ maps from $\mathbb{R}^{I(n_i)}$ to either $\mathbb{B}$ or $\mathbb{R}$ and $I(n_i)$ is the number of incoming edges to n_i. The network itself computes the class of functions $\mathcal{F}_{FN} : \mathbb{R}^n \to R$ where the range R is either $\mathbb{B}$ or $\mathbb{R}$.*

Definition 4.18 *A feedforward network of* Linear Threshold Elements *(LTEs) is a feedforward network in which every computation node computes a linear threshold function. Similarly, a feedforward network of* Linear Sigmoid Functions *is a feedforward network in which every computation node computes a function of the form $s[w_0 + w_1x_1 + \cdots + w_{I(n_i)}x_{I(n_i)}]$ where s is a sigmoid, for example of the form s_1 or s_2 defined in chapter 2.*

Results have been found concerning the VC dimensions of various types of feedforward network of LTEs. For completely general networks of this type, we have the following result which is due to Baum and Haussler [79].

Theorem 4.19 *Let $\mathcal{F}$ be the class of functions computed by a feedforward network of LTEs having $N \geq 2$ computation nodes and a total of W weights. The VC dimension of $\mathcal{F}$ is bounded above as follows.*

$$\mathcal{V}(\mathcal{F}) \leq 2W \log_2(eN). \tag{4.30}$$

Also, for $k \geq W$,

$$\triangle_{\mathcal{F}}(k) \leq \left(\frac{Nek}{W}\right)^W. \tag{4.31}$$

It was for some time an open question whether the logarithmic term in equation 4.30 was in fact necessary. Maass [121, 122] has recently demonstrated that certain networks of this type have a VC dimension of $\Omega(W \log_2 W)$, answering this question in the affirmative, and this result has important consequences in the following work.

For networks having a single hidden layer we have a further result due to Baum [123].

Theorem 4.20 *Let $\mathcal{F}$ be the class of functions computed by a fully connected feedforward network (there are no direct connections from inputs to the output node) having n inputs and a single hidden layer of h nodes. Then,*

$$\mathcal{V}(\mathcal{F}) \geq 2\left\lfloor\frac{h}{2}\right\rfloor n \tag{4.32}$$

where $\lfloor x \rfloor$ is the floor function, *giving the largest integer not greater than $x \in \mathbb{R}$.*

Various other lower bounds of $\Omega(W)$ on the VC dimension for different types of feedforward network of LTEs have been proved by Bartlett [124]. In this work, the result in theorem 4.20 is improved slightly to $\mathcal{V}(\mathcal{F}) \geq nh + 1$. Also, it is shown that if this type of network is not fully connected, but each hidden node is connected to the output node and hidden node i is connected to n_i inputs then $\mathcal{V}(\mathcal{F}) \geq \sum_{i=1}^{h} n_i + 1$. Bartlett also further generalizes the result to allow direct connections from inputs to output by showing that for a single hidden layer network in which inputs have c connections to other nodes, $\mathcal{V}(\mathcal{F}) \geq c + 1$. Finally, it is shown that if $\mathcal{F}$ is the class of functions computed by a fully connected feedforward network of LTEs having n inputs, h_1 nodes in hidden layer 1, h_2 nodes in hidden layer 2 and a single output, then $\mathcal{V}(\mathcal{F})$ is still $\Omega(W)$ provided h_2 is not too large. Again, we do not allow connections to 'skip' a layer, and so there are no direct connections from inputs to output etc. Specifically, if $n \geq h_1$ we have,

$$\mathcal{V}(\mathcal{F}) \geq nh_1 + h_1 \left[\min \left(h_2, \frac{2^{h_1}}{h_1^2/2 + h_1/2 + 1} \right) - 1 \right] + 1; \tag{4.33}$$

if $1 < n < h_1 \geq h_2$ we have,

$$\mathcal{V}(\mathcal{F}) \geq nh_1 + \frac{h_1(h_2 - 1)}{2} + 1; \tag{4.34}$$

and if $1 < n < h_1 < h_2$ we have,

$$\mathcal{V}(\mathcal{F}) \geq nh_1 + n \left[\min \left(h_2, \frac{\sum_{i=0}^{n} C_i^{h_1}}{h_1^2/2 + h_1/2 + 1} \right) - 1 \right] + 1, \tag{4.35}$$

where $C_i^{h_1}$ is the usual binomial coefficient.

We know that a Φ-net with W weights has VC dimension $\mathcal{V}(\mathcal{F}_n^{\Phi}) \leq W$, and that for many standard types of Φ-net we can expect to obtain $\mathcal{V}(\mathcal{F}_n^{\Phi}) = W$. We now assume that we have a Φ-net for which $\mathcal{V}(\mathcal{F}_n^{\Phi}) = W$ and compare this VC dimension with that for a typical feedforward network of LTEs.

Consider a feedforward network N of the type presented in theorem 4.20. We know that this network has a VC dimension $\mathcal{V}(\mathcal{F})$ where,

$$\mathcal{V}(\mathcal{F}) \geq 2 \left\lfloor \frac{h}{2} \right\rfloor n. \tag{4.36}$$

The network also has W_N weights where,

$$W_N = (n + 2)h + 1 = nh + 2h + 1. \tag{4.37}$$

Consequently, for large n and h, we have $W_N \simeq 2\lfloor \frac{h}{2} \rfloor n$. Comparing the two alternative VC dimensions, we see that the *upper* bound on $\mathcal{V}(\mathcal{F}_n^{\Phi})$ is approximately equal to the *lower* bound for $\mathcal{V}(\mathcal{F})$ when the feedforward network is large and the two networks have equal numbers of weights. In other words, the maximum capacity of the Φ-net is approximately equal to the minimum capacity of the feedforward network. Similar comparisons can be made using Bartlett's further results for single hidden layer networks, although comparisons for two hidden layer networks are more problematic.

In order for this comparison to be convincing at present, we must consider feedforward networks with many inputs and hidden nodes. However, we suspect that this is because the theorem due to Baum [123] from which the bound of theorem 4.20 is obtained actually states that if the network has $\lceil \frac{k}{n} \rceil$ hidden nodes then it can compute arbitrary dichotomies of *any* set of k points in general position, and not just the *single* set of k points which is required in order to obtain a VC dimension equal to k, and this suggests that the lower bound on the VC dimension for the feedforward network is not tight. Also, it is possible that the VC dimension of the feedforward network could in fact be larger than that of the Φ-net in general even though they have equal numbers of weights; this observation is supported by the fact that for *general* feedforward networks of LTEs with W' weights we have the upper bound,

$$\mathcal{V}(\mathcal{F}) \leq 2W' \log_2(eN) \tag{4.38}$$

of theorem 4.19 which will generally be *significantly* greater than $\mathcal{V}(\mathcal{F}_n^\Phi)$, although we must bear in mind that it is in general an overestimate.

We wish to show that feedforward networks of LTEs can have more capacity than Φ-nets with the same number of weights. Although the above observations are suggestive of this result, they do not provide a *definite* example of a network having more VC dimension per weight than a Φ-net with fixed basis functions. In order to construct some definite examples we use the results of Maass alluded to above, and a further result due to Sontag [125].

Theorem 4.21 (Maass [121]) *For arbitrary positive integers n there exists a feedforward network with n inputs and at most $33n$ edges, constructed using step, piecewise linear or sigmoid activation functions, which can shatter a set of $n \log_2 n$ vectors in $\{0,1\}^n$.*

Although this theorem allows us to construct a network for which $\mathcal{V}(\mathcal{F}) > W$, the catch is that the network must have a *very* large number of inputs. To see this, note that a network with $33n$ edges can have at most $66n$ nodes so the number of weights is at most about $100n$. We then require $\mathcal{V}(\mathcal{F}) \geq n \log_2 n > 100n \geq W$ which suggests that $n \simeq 2^{100}$. A much more realistic example can be obtained directly using the following result.

Theorem 4.22 (Maass [122]) *For arbitrary numbers of weights it is possible to construct feedforward networks of LTEs*[4] *using inputs in $\{0,1\}^n$ and having two hidden layers and at most $33W$ edges, which have VC dimension at least $W \log_2 W$.*

Furthermore, Sontag [125] constructs a class of 2-input feedforward networks, having a single hidden layer and direct connections between the inputs and the output node, for which $\mathcal{V}(\mathcal{F}) \geq W$.

4.5 Discussion

4.5.1 Results on the Growth Function and VC Dimension

We can immediately make two important observations relating to our results for growth functions and VC dimensions. Firstly, the number of variable weights in a network is often regarded as a suitable approximation to the VC dimension when no better estimate is available, and we have shown that in the case of Φ-nets with fixed basis functions this approximation is often in fact exactly correct. Note however that, as we demonstrated above and as we discuss further below, this approximation will not necessarily be correct for other types of network. Secondly, in the cases where we have derived bounds on the growth function or VC dimension, rather than exact values, the bounds are tight (with the possible exception of those given in corollary 4.15 and subsection 4.3.3).

4.5.2 Comparing the Capacities of Different Networks

Although the above comparison suggests that in general Φ-nets will require more weights than feedforward networks when applied to the same problem, it is important to remember when interpreting this result that the VC dimension is a rather general measure of capacity. In particular,

[4]Maass in fact constructs networks using LTEs which have outputs in $\{0,1\}$ rather than $\mathbb{B}$. This difference is irrelevant in making this comparison because the VC dimension of a Φ-net which produces outputs in $\{0,1\}$ is identical to that of one which produces outputs in $\mathbb{B}$.

it is important to remember that *specific* problems exist which particular Φ-nets can solve using *fewer* weights than feedforward networks of LTEs; a simple example is given in chapter 6.

Our comparison has concentrated on the use of feedforward networks of LTEs, whereas it is more usual for feedforward networks to be constructed using linear sigmoid functions. In using linear sigmoid functions we can only *increase* the power of a feedforward network, as such a network can compute any dichotomy computable by a feedforward network of LTEs with the same architecture [124]. Furthermore, results due to Sontag [125] suggest that we may be able to increase the VC dimension of a feedforward network by employing linear sigmoid functions rather than LTEs.

It would be useful to attempt a similar comparison of Φ-nets with alternative networks using different measures of capacity, such as that due to Cover [17]. Cover's results apply to a class of networks similar to Φ-nets, and have recently been extended to feedforward networks by Budinich and Milotti [126].

Note that our comparison suggests that one should be careful in equating the capacity of a network directly with the number of its weights. We have in mind some comments made by Blum and Rivest [23], who show that a simple multilayer network, the training task for which is **NP**-complete, can be replaced by a Φ-net which is at least as powerful (computes at least the same class of functions) but which can be trained in polynomial time. These authors point out that whereas the former network has $O(n)$ weights for n inputs, the latter network requires $O(n^2)$ weights; they then argue that consequently the number of training examples required by the latter network must be increased to provide equivalent generalization. Although this observation is entirely correct in spirit, as the latter network has at least the same capacity as the former, our comparison suggests that considering the increase in the number of weights alone could be misleading.

Finally, some recent results in approximation theory can be used to obtain the same conclusions as we obtained as a result of our comparison, for multilayer networks with a single hidden layer, using entirely different techniques (Sontag [127]).

4.6 Conclusion

In this chapter we have studied the growth function and the VC dimension of various Φ-nets. We began by addressing the completely general class of Φ-nets, obtaining in this case tight upper bounds on both quantities and giving a sufficient condition under which the VC dimension is exactly equal to its upper bound. This condition is important because it shows that the VC dimension of a Φ-net will often be exactly equal to the number of its variable weights; this is often assumed to be the case in practice when no exact value for the VC dimension is known. We then studied the VC dimension in the case of some restricted forms of the most general Φ-nets, and we discussed the way in which the present formalism can be used in order to study the use of self-structuring training algorithms. We then studied in detail the VC dimensions of radial basis function networks having both fixed and adapting centres, obtaining further exact results and bounds. Finally, we used the idea that the VC dimension can be regarded as a measure of the capacity of a network to propose an explanation for the observation that Φ-nets can in practice require a relatively large number of weights.

Chapter 5

The Generalization Ability of Φ-Nets II: Network Size and the Number of Training Examples

5.1 Introduction

Perhaps the best known, and most frequently cited, theoretical analysis of generalization to date is that of Baum and Haussler [79], based on the extended PAC learning formalism introduced in chapter 3. This analysis is aimed specifically at multilayer feedforward networks of LTEs, that is, networks in the style of the multilayer perceptron. The primary purpose of this chapter is to extend the analysis to the class of Φ-nets with fixed basis functions. In using the extended PAC learning formalism, we will be using a definition of generalization which can be stated in an intuitive sense as follows:

Definition 5.1 (Generalization (intuitive)) *Consider a connectionist network which has been satisfactorily trained using a sequence of training examples for a particular problem. If there is a* 'high enough' probability *that the actual error of the network for future examples drawn from the* same problem *is* 'small enough' *then we say that the connectionist network* generalizes.

As usual, we assume that both the training and testing examples are drawn from the same, arbitrary distribution P' on $\mathbb{R}^n \times \mathbb{B}$. Of course, by using this particular formalism in our analysis we are again limited to discussing networks having a single output, taking two distinct values, and hence to discussing problems with two classes.

We have already seen that in general the computational complexity of the training process for a Φ-net with fixed basis functions is smaller than that for a comparable multilayer perceptron, and that there are fundamental and significant differences between the two types of network. This chapter therefore also attempts, using the results obtained, to address the obvious question of how the generalization performance of a Φ-net with fixed basis functions compares to that of a comparable MLP. To our knowledge no significant attempt has previously been made to approach this question either experimentally or theoretically. The initial analysis of Φ-nets presented in this chapter closely follows that of [79] in order to facilitate a comparison of the two types of network.

In section 5.2 we derive necessary and sufficient conditions on the number of training examples required in training a Φ-net with fixed basis functions and W weights such that we can expect a particular generalization performance. We then briefly discuss the way in which these results should be interpreted, and we address the question of whether adaptation of the basis functions can be introduced without invalidating our results or increasing training time by an unacceptable amount. In section 5.3 we compare our results to similar results for feedforward networks of LTEs and in section 5.4 we use the idea of an l-restriction to approximately model the effect of the use of self-structuring training algorithms on our sufficient conditions. Section 5.5 discusses the way in which the comparative results should be interpreted, and suggests ways in which our results are open to improvement. It also shows how further relevant bounds can be derived, argues that further, experimental work is now desirable, and briefly discusses some alternative formalisms for the analysis of the generalization ability of Φ-nets. Section 5.6 concludes the chapter.

5.2 Generalization, Network Size, and the Number of Training Examples

In this section we use two theorems from extended PAC learning theory in conjunction with the results of chapter 4 in order to derive distribution independent upper and lower bounds on the number of training examples required in order to train a Φ-net with fixed basis functions such that it can be expected to deliver a given generalization performance. Subsection 5.2.1 deals with sufficient conditions and subsection 5.2.2 with necessary conditions.

In this chapter we assume that some measurability conditions on the classes of functions used are satisfied. These conditions are given in Blumer *et al.* [73] and Pollard [128] which should be consulted for further information. It is important to note that such conditions are extremely unlikely to cause any problem for practical connectionist networks (Haussler [129] and Anthony [130]).

5.2.1 Sufficient Conditions

In order to derive sufficient conditions we use the following theorem from extended PAC learning theory; this theorem is stated in the following form in [79] and follows from work presented in [73, 78].

Theorem 5.2 *Consider a class $\mathcal{F}$ of functions*[1] $f : \mathbb{R}^n \to \mathbb{B}$, *and a sequence T_k of k examples drawn independently according to some distribution P' on $\mathbb{R}^n \times \mathbb{B}$. Let γ and ϵ satisfy $0 < \gamma \leq 1$ and $0 < \epsilon$. Define $\mathcal{P}$ as the probability that there is a function $f \in \mathcal{F}$ which disagrees with at most a fraction $(1-\gamma)\epsilon$ of the examples T_k, but has error greater than ϵ. Then $\mathcal{P}$ satisfies the inequality,*

$$\mathcal{P} < 8\triangle_{\mathcal{F}}(2k)\exp\left(\frac{-\gamma^2\epsilon k}{4}\right). \tag{5.1}$$

This theorem is important because if we can obtain an upper bound on $\triangle_{\mathcal{F}}(i)$ then we can say something about the ability of a network that computes $\mathcal{F}$ to generalize. Specifically, if we can train the network to correctly classify a fraction $1-(1-\gamma)\epsilon$ of the k training examples, then the probability that its error — which is a measure of its generalization ability — is less than or equal to ϵ is at least $1-\mathcal{P}$. As $\triangle_{\mathcal{F}}(i)$ tends to depend quite specifically on the *size* of a network measured in terms of, for example, the total number of variable parameters, we can generally relate the size of a network to the number of examples which must be learnt.

[1]Note that this theorem also applies for environments X which are subsets of $\mathbb{R}^n$, for example $X = [0,1]^n$ or $X = \{0,1\}^n$ (see [73, 74]). In this case we consider functions $f : X \to \mathbb{B}$ and distributions P' on $X \times \mathbb{B}$.

Combining this theorem with the results of section 4.2, in particular the bound,

$$\triangle_{\mathcal{F}_n^\Phi}(i) \leq \triangle_{\mathcal{F}_m}(i) \tag{5.2}$$

derived in lemma 4.4, we can prove the following. Note that this theorem applies to *any* Φ-net having fixed basis functions, regardless of the actual form of the basis functions.

Theorem 5.3 *Consider a threshold Φ-net having n inputs and a set Φ of m basis functions. This network has $W = m+1$ weights and computes the class $\mathcal{F}_n^\Phi$ of functions; we assume that $W \geq 4$. Suppose we have a sequence T_k of k training examples drawn randomly from a distribution P' on $\mathbb{R}^n \times \mathbb{B}$, and some fixed value ϵ where $0 < \epsilon \leq \frac{1}{4}$. If $k \geq k_0 = \frac{32W}{\epsilon} \ln \frac{32}{\epsilon}$, and if it is possible to find a weight vector $\mathbf{w}$ which allows the Φ-net to correctly classify at least a fraction $1 - \frac{\epsilon}{2}$ of the training examples in T_k, then the probability (which we will call the* confidence*) that the actual error of the network is at most ϵ for future examples drawn from P' is at least $1 - 8\exp(-1.5W)$. If the number of training examples is $k \geq k_0 = \frac{64W}{\epsilon} \ln \frac{64}{\epsilon}$ then the probability is at least $1 - 8\exp(\frac{-\epsilon k}{32})$.*

Proof Application of lemma 4.4 and Sauer's lemma gives,

$$\triangle_{\mathcal{F}_n^\Phi}(k) \leq \triangle_{\mathcal{F}_m}(k) < \left(\frac{ek}{W}\right)^W \tag{5.3}$$

when $k \geq W$. We now let $\gamma = \frac{1}{2}$ and apply theorem 5.2. This gives,

$$\mathcal{P} < 8\left(\frac{2ek}{W}\right)^W \exp\left(\frac{-\epsilon k}{16}\right). \tag{5.4}$$

Now, note that an upper bound on $\mathcal{P}$ when $\gamma = \frac{1}{2}$ gives the required lower bound on the probability $1 - \mathcal{P}$ (the confidence) that a Φ-net trained to correctly classify all but a fraction $\frac{\epsilon}{2}$ of the training examples will have an actual error of at most ϵ for future examples. For the first proposed value of k_0, inserting the value of $k = k_0 = \frac{32W}{\epsilon} \ln \frac{32}{\epsilon}$ and re-arranging gives,

$$\mathcal{P} < 8\left(2e\frac{\epsilon}{32} \ln \frac{32}{\epsilon}\right)^W. \tag{5.5}$$

We now use an inequality from [79] which tells us that for $W \geq 4$, $N = 2$ and $\epsilon \leq \frac{1}{2}$,

$$8\left(2e\frac{\epsilon}{32N} \ln \frac{32N}{\epsilon}\right)^W < 8\exp(-1.5W). \tag{5.6}$$

Noting that this inequality still holds if we set $N = 1$ and restrict ϵ to the range $0 < \epsilon \leq \frac{1}{4}$, and comparing 5.6 with 5.5, we have,

$$\mathcal{P} < 8\exp(-1.5W) \tag{5.7}$$

which implies a confidence of at least $1 - 8\exp(-1.5W)$.

For the second proposed value of k_0, we use a further inequality from [79] which states that for $W \geq 4$, $N \geq 2$, and $k \geq \frac{64W}{\epsilon} \ln \frac{64N}{\epsilon}$,

$$\left(\frac{2Nek}{W}\right)^W \leq \exp\left(\frac{\epsilon k}{32}\right) \tag{5.8}$$

when $0 < \epsilon \leq \frac{1}{2}$. Again, restricting ϵ to the range $0 < \epsilon \leq \frac{1}{4}$, we obtain for $W \geq 4$, $N = 2$ and $k \geq \frac{64W}{\epsilon} \ln \frac{64}{\epsilon}$ the inequality,

$$\left(\frac{2ek}{W}\right)^W < \left(\frac{2Nek}{W}\right)^W \leq \exp\left(\frac{\epsilon k}{32}\right). \tag{5.9}$$

Thus, comparing 5.9 with 5.4 we have,

$$\mathcal{P} < 8 \exp\left(\frac{\epsilon k}{32}\right) \exp\left(\frac{-\epsilon k}{16}\right) = 8 \exp\left(\frac{-\epsilon k}{32}\right) \tag{5.10}$$

which implies a confidence of at least $1 - 8\exp\left(\frac{-\epsilon k}{32}\right)$. □

Recall that theorem 5.2 also applies for environments X which are subsets of $\mathbb{R}^n$, such as $X = [0,1]^n$. It is easily verified that our upper bound on $\triangle_{\mathcal{F}_n^\Phi}(i)$ also holds when $X \subseteq \mathbb{R}^n$, and consequently that theorem 5.3 also holds in this case.

5.2.2 Necessary Conditions

In order to derive necessary conditions we use the following theorem from extended PAC learning theory. Again, this theorem is stated in the following form in [79], and in this case follows from results in [73, 85].

Theorem 5.4 *Consider a class $\mathcal{F}$ of functions $f : \mathbb{R}^n \rightarrow \mathbb{B}$ for which the VC dimension is $\mathcal{V}(\mathcal{F}) \geq 2$. Let A be an algorithm for which the input is a sequence of labelled training examples, and which produces a function $g : \mathbb{R}^n \rightarrow \mathbb{B}$. Given any δ and ϵ where $0 < \delta \leq \frac{1}{100}$ and $0 < \epsilon \leq \frac{1}{8}$, and,*

$$k < \max\left[\frac{1-\epsilon}{\epsilon}\ln\frac{1}{\delta}, \frac{\mathcal{V}(\mathcal{F})-1}{32\epsilon}\right] \tag{5.11}$$

there exists $f \in \mathcal{F}$, and a distribution P' on $\mathbb{R}^n \times \mathbb{B}$ for which,

$$\Pr[(\mathbf{x}, a) \text{ such that } a \neq f(\mathbf{x})] = 0, \tag{5.12}$$

such that for a sample T_k of k examples chosen according to P', there is a probability of at least δ that the algorithm A produces a function with error greater than ϵ.

Using the various results derived in sections 4.2 and 4.3, providing lower bounds on the VC dimension for various Φ-nets, we can immediately state the following corollaries.

Corollary 5.5 *Consider a restricted Φ-net which has n inputs and $W = m + 1$ weights. Also, let $0 < \epsilon \leq \frac{1}{8}$. If any training algorithm for this Φ-net uses fewer than k training examples where,*

$$k = \frac{n}{32\epsilon} \tag{5.13}$$

then there exists some distribution P' such that there exists a weight vector $\mathbf{w}$ for the Φ-net which will correctly classify all inputs according to P'. However the probability of finding a $\mathbf{w}$ such that the error, defined as the probability according to P' that $f_{\mathbf{w}}(\mathbf{x}) \neq o$ where $(\mathbf{x}, o)$ is a random example, is in fact greater than ϵ is at least $\frac{1}{100}$.

Proof This is a direct consequence of lemma 4.7 and theorem 5.4 when we let $\delta = \frac{1}{100}$. □

Clearly we can easily obtain similar results for the more general restricted Φ-nets suggested in chapter 4. If we have a restricted Φ-net for which $\Phi = \{\Phi_1, \Phi_2\}$ where Φ_1 is the set of basis functions corresponding to an (n, d) discriminator then

$$\mathcal{V}(\mathcal{F}_n^\Phi) \geq \binom{n+d}{d} \tag{5.14}$$

and the necessary number of training examples, under the same conditions as in corollary 5.5, is,

$$k = \frac{\begin{pmatrix} n+d \\ d \end{pmatrix} - 1}{32\epsilon}. \tag{5.15}$$

Similarly, if we know the VC dimension of a Φ-net constructed using any particular set Φ_1 of basis functions then further, similar bounds are easily obtained for any Φ-net constructed using the set of basis functions $\Phi = \{\Phi_1, \Phi_2\}$ where Φ_2 is any further suitable set of basis functions.

As pointed out in section 4.2, it is not possible to derive a sensible lower bound on $\mathcal{V}(\mathcal{F}_n^\Phi)$ for completely general Φ-nets; it was the fact that such a lower bound is required in order to obtain necessary conditions using theorem 5.4 that motivated us to define restricted Φ-nets. Clearly, for any Φ-net having a VC dimension which meets the upper bound of lemma 4.4, that is, for which $\mathcal{V}(\mathcal{F}_n^\Phi) = W$, a further corollary can be obtained as follows.

Corollary 5.6 *Let $\mathcal{F}_n^\Phi$ be the class of functions computed by a Φ-net, where $\mathcal{V}(\mathcal{F}_n^\Phi) = W$. Under precisely the same conditions as in corollary 5.5 the minimum necessary number of training examples is at least,*

$$k = \frac{W-1}{32\epsilon}. \tag{5.16}$$

The value given in corollary 5.6 is valid, for example, in the case of an LDF or a PDF, and in the case of some of the radial basis function networks considered in the previous chapter. It is also valid for any Φ-net in which the set of functions $\{1, \phi_1, \ldots, \phi_m\}$ is linearly independent. However, this value will in some cases be an overestimate of the necessary number of examples.

In the case of the radial basis function networks considered in the previous chapter we know that $\mathcal{V}(\mathcal{F}) \geq p$ where p is the number of radial basis functions, and we also know that this lower bound applies whether the centres are fixed or adapting. We therefore have the following result.

Corollary 5.7 *For the different types of radial basis function network considered in corollaries 4.14 and 4.15, and under the same conditions as in corollary 5.5, the minimum necessary number of training examples is at least,*

$$k = \frac{p-1}{32\epsilon} \tag{5.17}$$

regardless of whether the centres $\{\mathbf{y}_i\}$ are fixed or adapting.

5.2.3 Interpretation of the Results

The results presented above are at first sight rather impenetrable, and we now try to give some further insight into their meaning. Table 5.1 summarizes the results obtained above on necessary and sufficient numbers of training examples.

Necessary Conditions

The necessary conditions tell us simply that, if we do not use enough training examples, then although a weight vector may exist which gives an error of zero, there is in this case a probability of at least $\frac{1}{100}$ that, no matter what training algorithm is used, the actual network produced has an error greater than ϵ. This is illustrated in figure 5.1.

Sufficient conditions for Φ-nets. $0 < \epsilon \leq \frac{1}{4}$		
Characteristics of Φ-net	Sufficient number of training examples	Confidence
W weights.	$k \geq k_0 = \frac{32W}{\epsilon} \ln \frac{32}{\epsilon}$	$1 - 8\exp(-1.5W)$
W weights.	$k \geq k_0 = \frac{64W}{\epsilon} \ln \frac{64}{\epsilon}$	$1 - 8\exp\left(-\frac{\epsilon k}{32}\right)$

Necessary conditions for Φ-nets. $0 < \epsilon \leq \frac{1}{8}$	
Characteristics of Φ-net	Necessary number of training examples
Restricted Φ-net with n inputs.	$k = \frac{n}{32\epsilon}$
General restricted Φ-net (having a subset of basis functions corresponding to those of an (n, d) discriminator).	$k = \frac{C_d^{n+d} - 1}{32\epsilon}$
Φ-net having W weights and such that $\mathcal{V}(\mathcal{F}_n^{\Phi}) = W$.	$k = \frac{W-1}{32\epsilon}$
RBFNs as described in corollaries 4.14 and 4.15.	$k = \frac{p-1}{32\epsilon}$

Table 5.1: Summary of Results on the Sufficient and Necessary Numbers of Training Examples Required Such That we Expect a Particular Generalization Performance when Using a Φ-Net

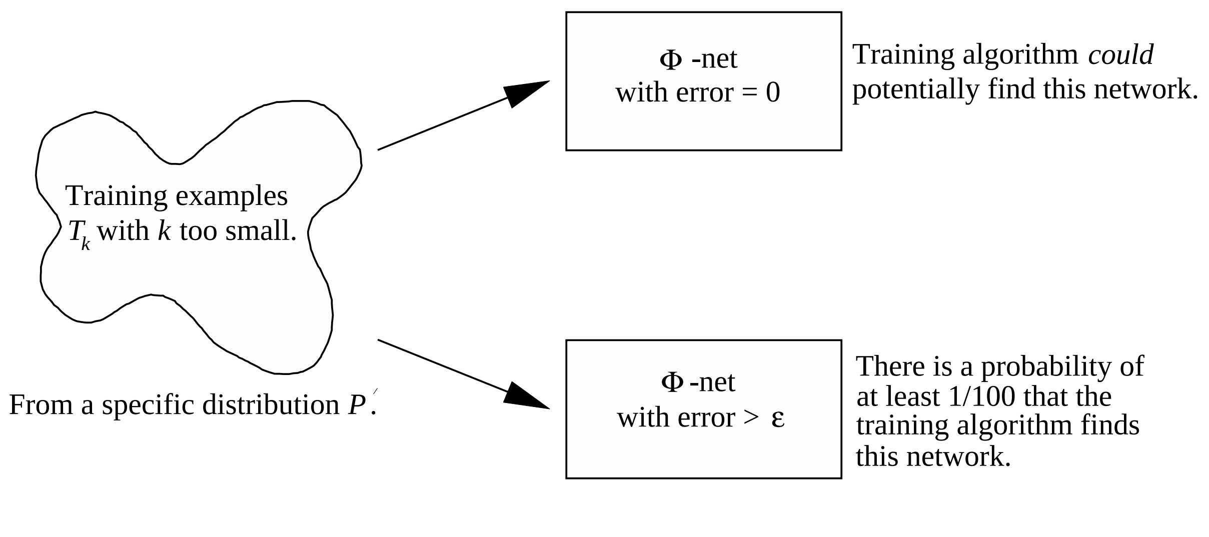

Φ -net
with error = 0
Training algorithm *could* potentially find this network.
Training examples T_k with k too small.
From a specific distribution P'.
Φ -net
with error > ε
There is a probability of at least 1/100 that the training algorithm finds this network.

Figure 5.1: Interpretation of the necessary conditions. Although a network exists which has zero error there is a finite probability that the training algorithm produces a network with error greater than ϵ.

In the light of the discussion in the previous chapter, we can expect that in general around $\frac{W-1}{32\epsilon}$ examples will be the minimum necessary. For a small error of $\epsilon = \frac{1}{32}$ this is approximately equal to the number of weights in the network, and so this result upholds and justifies one of the well known 'rules of thumb' regarding the training of connectionist networks: that one should use at least as many training examples as there are weights in the network. A similar observation was made in [79] relating to feedforward networks of LTEs.

Sufficient Conditions

Note that for all practical values of W, the confidence values given in table 5.1 are very close to unity. In chapter 3 we saw that there appears to be a direct relationship between the size of a network, measured in an appropriate manner, and its ability to generalize. The results of our analysis of sufficient numbers of training examples uphold this observation and can be interpreted in an intuitive sense as follows:

If it is possible to load enough training examples into a small enough network then there is a high probability of obtaining a given generalization performance.

Again, similar interpretations have been suggested for other networks by other authors. The meaning of the results is further illustrated in figure 5.2.

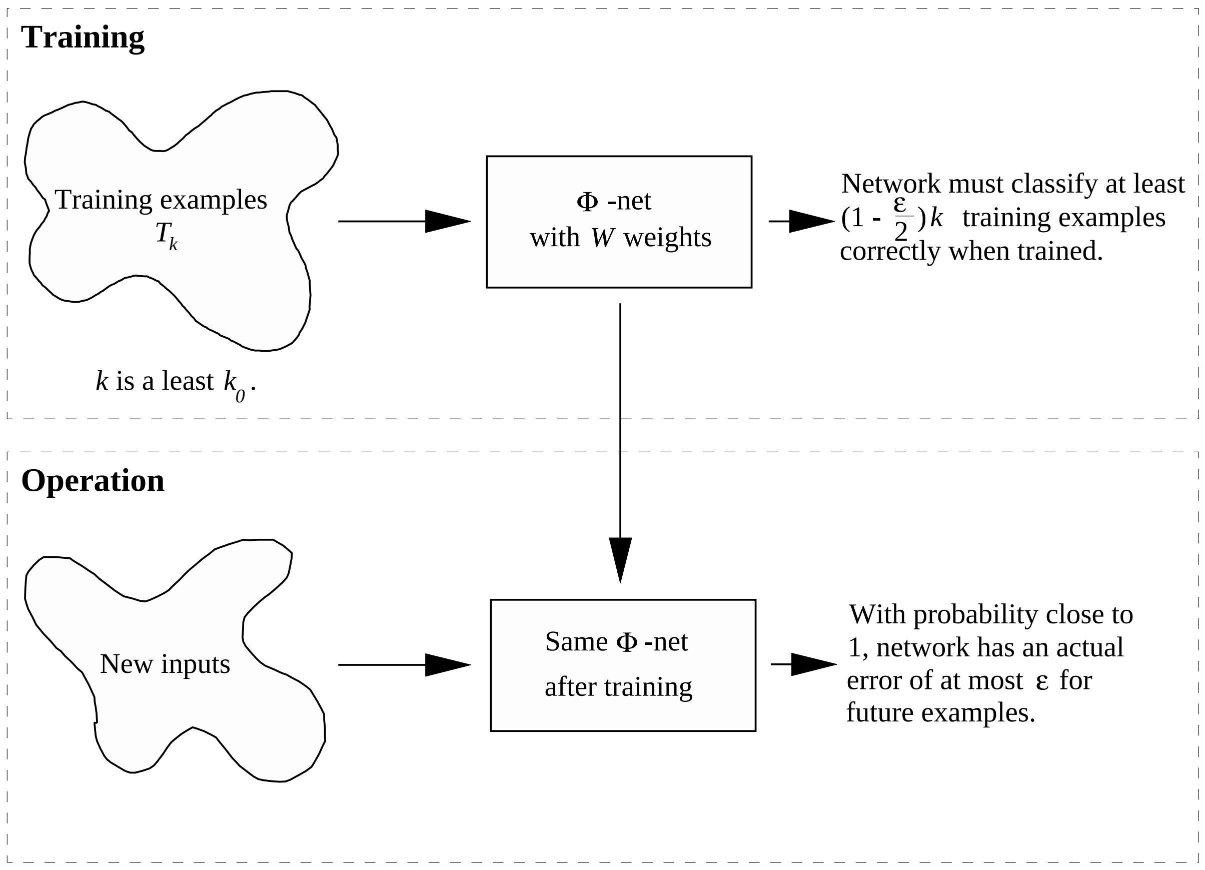
These conditions, although fully applicable bounds in their own right on the sufficient number of training examples, are not as easily interpreted in a practical sense as the necessary conditions. Inserting some realistic values for W and ϵ into the two expressions for k_0 immediately shows that extremely large sets of training examples are required in order to apply these results. For example, taking the condition,

$$k \geq k_0 = \frac{32W}{\epsilon} \ln \frac{32}{\epsilon} \tag{5.18}$$

and inserting values of W ranging from 10 to 500 and four different values of ϵ gives the values for k_0 illustrated in figure 5.3.

We thus see that independence to the distribution P' from which the training examples and the examples encountered during operation are drawn is very much a two edged sword. It has the obvious advantage of providing bounds which can be applied without requiring this particular piece of *a priori* knowledge about the environment in which the network operates to be specifically introduced. However the requirement that the bounds hold for *any* distribution, even those which may be highly unrealistic, leads to bounds which are perhaps unreasonably high. The available research in distribution-dependent learning, which we mentioned briefly in chapter 3, could be applied in order to overcome this problem, while of course weakening the results somewhat; we suspect that, in common with the observations made in [79], the logarithmic terms in our bounds may not be necessary if only realistic distributions are considered. However, if you *can* obtain and deal with enough training data, distribution independence is obviously a significant advantage.

There are two further reasons for the unwieldy nature of these bounds, and with further research it may be possible to significantly improve the situation. Firstly, recall that we have deliberately produced bounds comparable to those in [79] in order to facilitate a comparison with an alternative type of network. However, it is possible to improve the constants used, and we discuss this further in section 5.5. Unfortunately, the improvements possible at present still do not lead to truly practical bounds. Secondly, the bounds apply regardless of the precise characteristics of the algorithm used to train the network. In particular, they will apply in the worst possible case, in which the algorithm picks a function in $\mathcal{F}_n^\Phi$ that correctly classifies the



Training
Training examples T_k
k is a least k_0.
Φ -net with W weights
Network must classify at least $(1 - \frac{\varepsilon}{2})k$ training examples correctly when trained.
Operation
New inputs
Same Φ -net after training
With probability close to 1, network has an actual error of at most ε for future examples.

Figure 5.2: Interpretation of the sufficient conditions. If it is possible to learn enough training examples using a small enough network then there is a high probability that future performance achieves a specific accuracy.

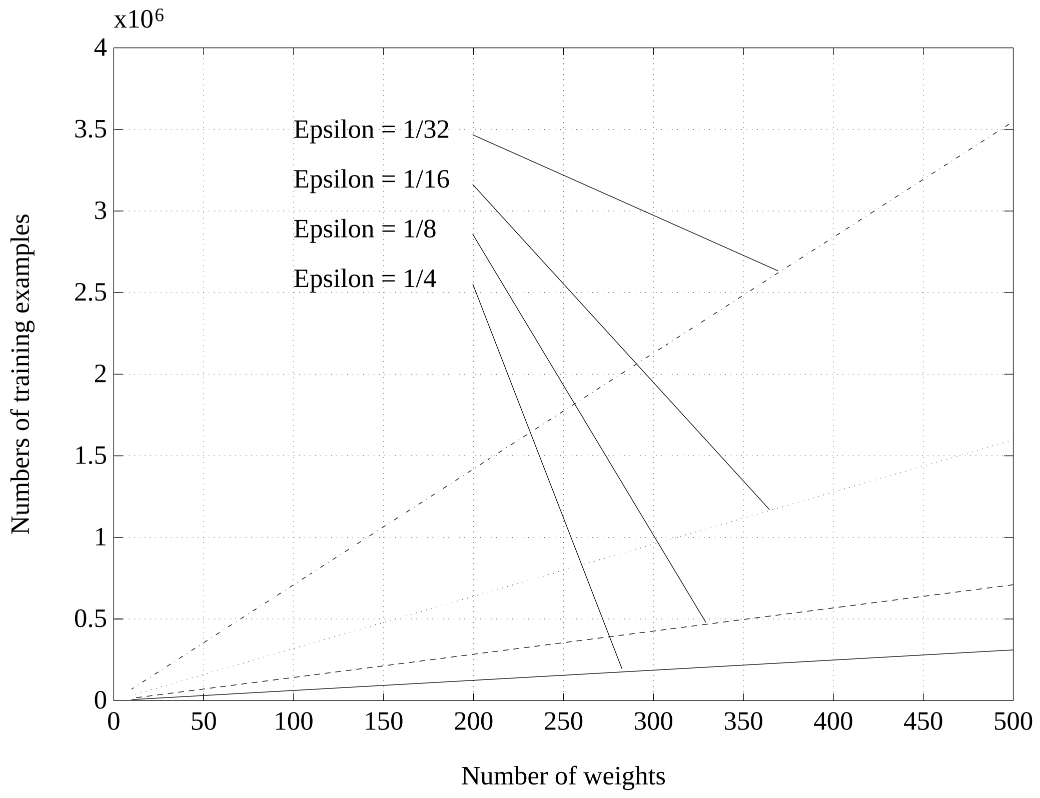


Figure 5.3: Illustration of the sufficient numbers k_0 of training examples, given by one of the bounds derived, for networks having different numbers W of weights and for different values of ϵ.

relevant number of training examples but has the worst possible performance on future examples (see Haussler *et al.* [92]). This is not a situation that we would reasonably expect to occur in practice. The further improvement of our upper bounds provides an important potential area for continuing this research.

5.2.4 Choosing a Set of Basis Functions: Can We Cheat Theorem 5.3?

Theorem 5.3 applies specifically to the case of fixed basis functions. If we can find a Φ-net with fixed basis functions and with the appropriate number of weights, which correctly classifies the required number of training examples, then we can predict with high probability its performance on new examples. As we mentioned in chapter 2, the use of fixed basis functions has been criticized for being too restrictive, although we argued that this is not necessarily the case and that there are significant advantages in not allowing basis functions to adapt. Also, recall that we mentioned training algorithms, in particular one due to Moody and Darken [30], which appear to allow us to introduce adaptation of basis functions while retaining the major advantage (fast training) associated with fixed basis functions. We now discuss the use of adapting basis functions in the context of the preceding work.

Firstly, note that, in the context of the preceding work in this chapter, we cannot directly introduce any scheme which adapts the basis functions at the same time as the weight vector $\mathbf{w}$ using nonlinear optimization, as the overall network is then likely to be much more flexible than in the fixed basis function case and can therefore be expected to have a significantly larger VC dimension. The larger VC dimension in turn increases the minimum sufficient number of training examples (see the proof of theorem 5.3). Also, the need for nonlinear optimization makes training significantly more difficult and time consuming, and this is something that we want to avoid.

What has not previously been appreciated however is that it appears that we can use a simple technique allowing us to adapt the basis functions without invalidating the bounds of theorem 5.3 applying to the case of fixed basis functions. The essence of the technique is simply that we adapt basis functions entirely separately from the weight vector $\mathbf{w}$, and we shall call training algorithms which use this technique *hybrid training algorithms*. In the following we use radial basis function networks as a source of examples, however it is important to remember that hybrid training algorithms can in principle be devised for any Φ-net in which the basis functions are parameterized. In the remainder of this subsection we first provide a theoretical, and then an intuitive argument in favor of the use of hybrid training algorithms. Our theoretical argument is based on the application of theorem 5.3, which is itself derived in part using theorem 5.2. The reader should note that the question of whether theorems 5.2 and 5.3 can strictly be applied in the manner suggested in the remainder of this subsection has attracted controversy, and a further investigation into the extent to which our theoretical argument is valid consequently forms an important, and potentially rich area for future research.

To begin with a very simple example, consider the standard technique for choosing radial basis function centres which we summarized in chapter 2: fixed centres are chosen corresponding to randomly chosen training examples. This provides us with a Φ-net with fixed basis functions to which theorem 5.3 can be applied; note however that because this technique for choosing centres depends directly on the training data it can be regarded as a very simple method for adapting the basis functions. The important difference between this type of adaptation and the type which is not compatible with theorem 5.3 is that, in the former, adaptation of basis functions takes place entirely separately from adaptation of $\mathbf{w}$ such that the basis functions can be fixed before adaptation of $\mathbf{w}$ begins.

As a more realistic example, consider the algorithm reviewed briefly in chapter 2 (subsection 2.5.3) and originally introduced by Moody and Darken [30]. This algorithm applies to networks of the form,

$$f_{\mathbf{w}}(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x}) \tag{5.19}$$

where,

$$\phi_i(\mathbf{x}) = \phi\left(\frac{\|\mathbf{x} - \mathbf{y}_i\|}{\sigma_i}\right) \tag{5.20}$$

and the weights w_i, centres $\mathbf{y}_i$ and widths σ_i are variable. It allows the values of $\mathbf{y}_i$ and σ_i to be chosen and fixed using unsupervised techniques, thus producing a Φ-net with fixed basis functions (provided we add a term w_0 to equation 5.19), before the weight vector $\mathbf{w}$ is found using supervised training. We can thus in principle take a network size and the corresponding required number of training examples from theorem 5.3, and use unsupervised methods to obtain a Φ-net with fixed basis functions and the appropriate number of weights such that the basis functions are well matched to the structure of the training examples. We then train the resulting network by adapting $\mathbf{w}$ as usual in an attempt to learn the training examples.

The precise difference between this type of two-stage, hybrid training algorithm and the alternative approach using global nonlinear optimization is, for the purposes of analysing generalization ability, quite subtle. In the latter case, we use the entire class of functions available in order to learn the required number of examples and this leads to the problems mentioned (namely, increased training time and increased sufficient number of training examples for generalization in the extended PAC learning framework). In the former case we use our knowledge of the available training examples to discard many of these functions — the ones which do not seem to be appropriate — and use the resulting reduced class of functions to learn the training examples. We therefore appear to be reducing the *capacity* of the class of functions available, before we attempt to adapt $\mathbf{w}$. (In one sense therefore, the procedure can be regarded as similar to self-structuring.) The procedure also has the major advantage that it is in general considerably faster than global nonlinear optimization.

In the consideration of generalization performance, these arguments justify the use of hybrid training algorithms, such as that of Moody and Darken, using theorem 5.3. However, it is interesting to ask whether our conclusion — that the use of hybrid training algorithms may lead to a reduction in the number of training examples required to obtain valid generalization, when compared with the number required by training algorithms using global nonlinear optimization — is likely to be correct in general. We conjecture that this conclusion will hold in practice for the following reasons. Consider the intuitive explanation for the importance of capacity introduced in chapter 3. Using this intuitive framework we would expect the conclusion to hold if the first step of a hybrid training algorithm discards functions in such a way that:

1. During subsequent adaptation of $\mathbf{w}$ the probability that the network can compute a function which performs well on the training data is not significantly reduced.
2. The class of functions which the network can compute that perform well on the training data *is* reduced, but in such a way that those providing 'good' generalizations are retained.

Now, provided we are careful in devising the first step of a hybrid training algorithm, in which we fix the basis functions, we can reasonably expect that both of these requirements will be met. To see this in a specific case[2], consider Moody and Darken's algorithm, in which we use basis functions $\phi_i(\mathbf{x}) = \exp\left[-\frac{\|\mathbf{x}-\mathbf{y}_i\|^2}{\sigma_i^2}\right]$, which is the type of basis function used in the experiments

[2]The following example is most applicable if we consider a network with a real-valued output, however similar examples can be constructed for the case of other types of output, such as ± 1.

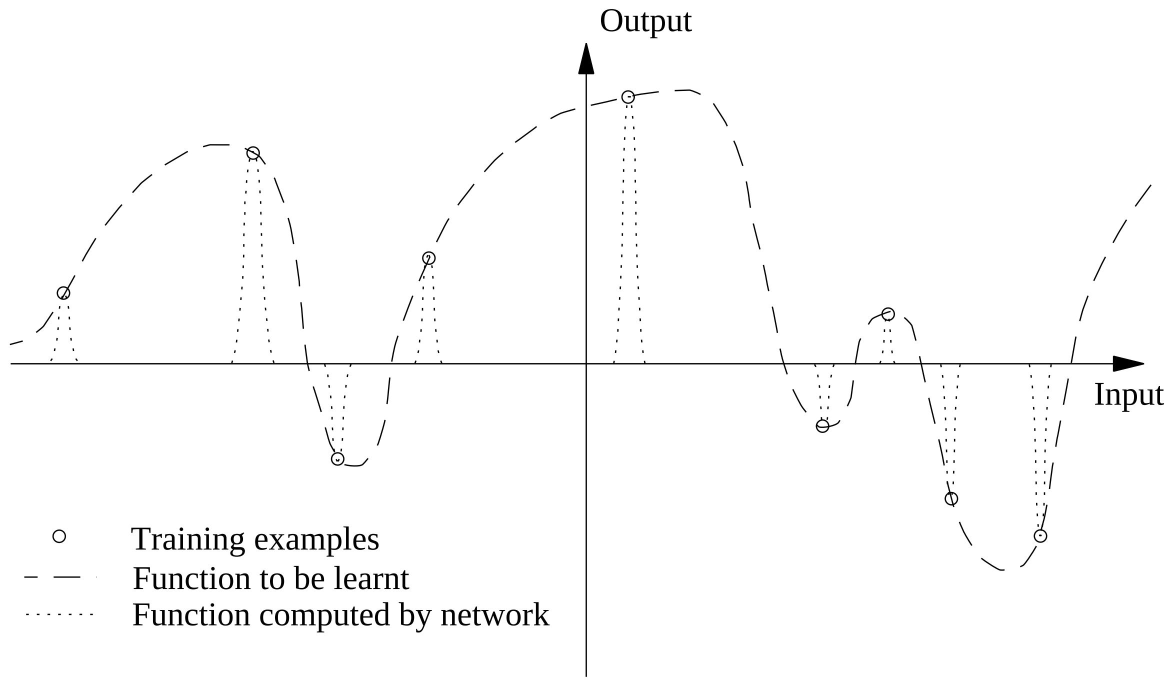


Figure 5.4: A function which performs well on the training examples but which generalizes badly. This type of function will be discarded by a suitable hybrid training algorithm before adaptation of **w** begins.

in [30]. Assume that we have a set of training examples, and a network with a sufficient number of basis functions which we attempt to train using simultaneous nonlinear optimization of all available parameters. The class of functions available to the network is likely to contain functions which perform well on the training examples but which generalize badly; for example, we may be able to construct a function which performs well on the training data by making centres $\mathbf{y}_i$ correspond to the examples, making all values of σ_i very small, and setting the remaining weights appropriately. This function would be unlikely to perform well in general on new examples; this is illustrated in figure 5.4 for a network with a single input. However, if Moody and Darken's hybrid training algorithm (or a suitable alternative) is applied, we discard functions such as this when the basis functions are fixed (because the σ_i are chosen more sensibly[3]) while retaining many other functions which perform well on the training examples.

This argument is clearly an intuitive one, and is at present quite tentative; it has been included primarily as motivation for future research, as hybrid training algorithms appear to be an excellent approach to training connectionist networks (this is supported by the experimental results reported in [30]). Clearly, a more rigorous, theoretical investigation would be desirable. Similarly, we know of no practical study to date that attempts to verify our suggestions, and this also provides an open area for further research.

Although this technique could in principle be extended to apply to feedforward networks of LTEs or linear sigmoid functions — it corresponds to fixing a subset of the available weights to values which we think will be appropriate — we know of no practical training algorithm which allows us to do so. (Note that we can regard the training of a feedforward network of LTEs or

[3]The simplest method suggested in [30] for setting σ_i is to make all values σ_i equal to the average distance between centres and their nearest neighbour.

Sufficient conditions for feedforward networks of LTEs with graph G and $0 < \epsilon \leq \frac{1}{2}$		
Characteristics of network	Sufficient number of training examples	Confidence
W weights, $N \geq 2$ LTEs,	$k \geq k_0 = \frac{32W}{\epsilon} \ln \frac{32N}{\epsilon}$	$1 - 8\exp(-1.5W)$
E edges.	$k \geq k_0 = \frac{64W}{\epsilon} \ln \frac{64N}{\epsilon}$	$1 - 8\exp\left(-\frac{\epsilon k}{32}\right)$

Table 5.2: Summary of Results due to Baum and Haussler on the Sufficient Number of Training Examples Required to Effectively Guarantee a Particular Generalization Performance using a Feedforward Network of LTEs

linear sigmoid functions as analogous to the global nonlinear optimization case for Φ-nets.) This is probably a result of the fact that the local nature of the actual basis functions used in many Φ-nets allows them to be tailored in an obvious manner to fit the structure of the available training examples, whereas it is more difficult to see how this could be done for feedforward networks of LTEs or linear sigmoid functions. The *Cascade-Correlation* technique introduced by Fahlman and Lebiere [131] is perhaps the technique closest in spirit to a hybrid training algorithm for MLP type networks.

5.3 Comparison of Φ-Nets with Feedforward Networks of LTEs

Table 5.2 summarizes the sufficient conditions derived in [79] for feedforward networks of LTEs having N computation nodes. These are equivalent to the set of sufficient conditions we have derived for Φ-nets, which were summarized in table 5.1, in the sense that they tell us the sufficient number of training examples such that if a fraction $1 - \frac{\epsilon}{2}$ of the training sequence is correctly classified then the confidence that the actual error of the network is at most ϵ for future examples is as stated. In this section we attempt to compare the generalization ability of Φ-nets to that of, in general multilayer, feedforward networks of LTEs, and hence address one of the questions raised in the introduction to this chapter: can we expect to obtain superior generalization performance if we use a Φ-net in preference to a comparable feedforward network of LTEs in order to solve a problem? We will consider two networks to be comparable simply if they have the same number of variable weights; this assumption is discussed further below.

5.3.1 A Simple Comparison

We begin by making a simple comparison between the two different types of network, assuming that in preference to a feedforward network of LTEs we use a Φ-net with the same number of weights. We also assume that each network is capable of learning to correctly classify the necessary fraction $(1 - \frac{\epsilon}{2})$ of the relevant number of training examples; both of these assumptions are discussed in the next subsection. A conclusion which can immediately be drawn from the sufficient conditions summarized in tables 5.1 and 5.2 is that, under these assumptions, the current bounds suggest that considerably fewer training examples are required by the Φ-net than by the feedforward network in order to effectively guarantee[4] the same generalization performance.

To give a specific example, if we use networks with $W = 200$ weights, a feedforward network having $N = 50$ computation nodes, and a value of $\epsilon = \frac{1}{4}$, the sufficient number of training

[4]Note that the confidence values are very close to unity for realistic values of W. For example, if $W = 10$ we have a confidence of $1 - 8\exp(-15) = 0.999998$. A similar observation applies when the confidence is $1 - 8\exp\left(-\frac{\epsilon k}{32}\right)$.

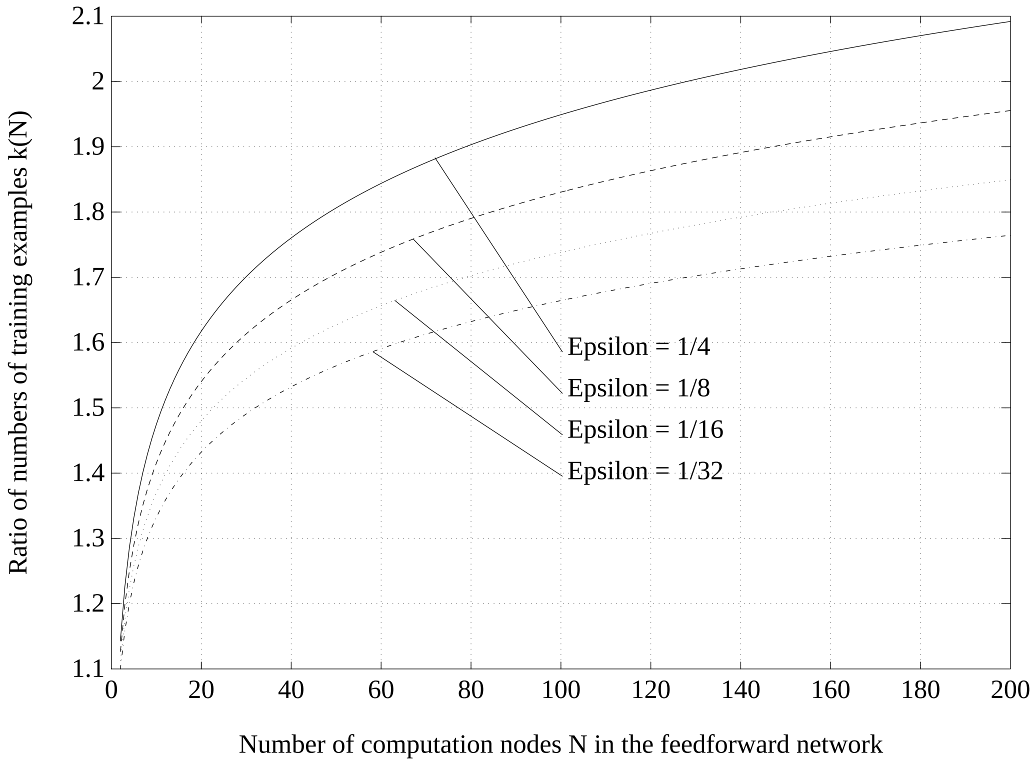


Figure 5.5: Comparison of sufficient conditions for Φ-nets with those for feedforward networks of LTEs. All networks have $W = 500$ weights. The quantity $k(N)$ is defined in full in the text.

examples for a Φ-net is,

$$k_0^{(1)} = \frac{32W}{\epsilon} \ln \frac{32}{\epsilon} \simeq 124,000 \tag{5.21}$$

whereas that for the feedforward network is,

$$k_0^{(2)} = \frac{32W}{\epsilon} \ln \frac{32N}{\epsilon} = k_0^{(1)} + \frac{32W}{\epsilon} \ln N \simeq k_0^{(1)} + 100,000 \tag{5.22}$$

and the difference between $k_0^{(1)}$ and $k_0^{(2)}$ clearly increases as we impose a more rigorous accuracy requirement by decreasing ϵ. In both cases we obtain the same confidence.

Figure 5.5 provides a further comparison, plotting $k(N) = (k_0^{(2)}/k_0^{(1)})$ for $W = 500$, different values of N and four different values of ϵ. Clearly, for the cases of interest, in which the feedforward network has at least one hidden layer and hence N is likely to be significantly greater than 2, the use of a Φ-net leads to a significant reduction in the sufficient number of training examples. Note that in plotting $(k_0^{(2)}/k_0^{(1)})$ we cancel the leading constants. This is highly desirable because, as we have already observed, they are likely to be larger than necessary. The same comment applies to the graphs presented in figure 5.6 in the next section.

A similar conclusion can be drawn using the alternative sufficient conditions,

$$
\begin{array}{lll}
k_0^{(1)} = \frac{64W}{\epsilon}\ln\frac{64}{\epsilon} & & \text{(for }\Phi\text{-nets)} \\
k_0^{(2)} = \frac{64W}{\epsilon}\ln\frac{64N}{\epsilon} & & \text{(for feedforward networks of LTEs)}
\end{array}
$$

if we make the reasonable assumption that the difference in the confidence for the two cases is negligible.

5.3.2 Discussion: How Comparable are Comparable Networks?

In the above comparison we made the assumption that the two networks have equal numbers of variable weights; we introduced this assumption because it is the obvious way in which to make two networks 'comparable'. Now, note that in order to apply the bounds used above, we must first be able to load the networks with a specific number of training examples. A possible problem with the above comparison is that it effectively assumes that the two different types of network have comparable power, in terms of their ability to learn to classify the required number of training examples to a sufficient accuracy, as a result of the fact that we have given them equal numbers of weights. We must obviously address the validity of this assumption.

In the light of the results presented in chapter 4 we suspect that in general the Φ-net can be expected to be less likely than the feedforward network of LTEs to be capable of learning some given sequence of training examples. Note however that the Φ-net is at an advantage in one respect, because the number of training examples that it has to learn such that we can apply the bounds is *less* than that for the feedforward network of LTEs. This complicates the situation somewhat, and the precise validity of the assumption that the two networks are equally powerful for the purposes of applying the bounds is not at present clear. Consequently, our second assumption — that each network *can* in fact learn to correctly classify a fraction $1-\frac{\epsilon}{2}$ of the relevant number of training examples — is clearly an important one. Whether this second assumption is valid will clearly be dependent on the specific characteristics of the networks used, and on the actual problem addressed.

5.4 The use of Self-Structuring

Recall that in chapter 4 we argued that self-structuring cannot be analysed directly using methods based on the growth function and VC dimension, and introduced the idea of the l-restriction of a Φ-net in order to allow us to obtain some insight into the effect of self-structuring. We now generalize the results of theorem 5.3 by obtaining a sufficient condition for networks which compute the class of functions $\mathcal{R}^{W'}(\mathcal{F}_n^{\Phi})$ rather than $\mathcal{F}_n^{\Phi}$, where $\mathcal{F}_n^{\Phi}$ is the class of functions computed by a Φ-net with $W \geq 5$ weights. We assume that $W' \geq 4$ and $W > W'$. In effect, although we are still using a network with W weights, we are now insisting that the training algorithm finds a weight vector with at most W' non-zero weights.

Theorem 5.8 *Let $\mathcal{R}^{W'}(\mathcal{F}_n^{\Phi})$ be a class of functions computed by a Φ-net as described. Then under the conditions described in theorem 5.3 we can obtain a confidence of* $1 - 8\exp(-1.5W')$ *using* $k \geq k_0 = \frac{32W'}{\epsilon}\ln\frac{32W}{\epsilon}$ *training examples.*

Proof We begin by restating equation 4.10 which allows us to write $\mathcal{R}^{W'}(\mathcal{F}_n^{\Phi})$ as,

$$
\mathcal{R}^{W'}(\mathcal{F}_n^{\Phi}) = \bigcup_{i=1}^{N(W',W)} \mathcal{F}_n^{\Phi_i} = \mathcal{F}_n^{\Phi_1} \cup \mathcal{F}_n^{\Phi_2} \cup \cdots \cup \mathcal{F}_n^{\Phi_{N(W',W)}} \tag{5.23}
$$

and which allows us to think of the W'-restriction in terms of a collection of smaller Φ-nets. Now, as in the proof of theorem 5.3, we have from lemma 4.4 and Sauer's lemma,

$$\triangle_{\mathcal{F}_n^{\Phi_i}}(k) \leq \triangle_{\mathcal{F}_{W'-1}}(k) < \left(\frac{ek}{W'}\right)^{W'} \tag{5.24}$$

for $k \geq W'$. Also, as the binomial coefficient obeys,

$$\binom{i}{j} = \binom{i-1}{j-1} + \binom{i-1}{j} \tag{5.25}$$

for $1 \leq j \leq i$ (see for example Biggs [119]) we have,

$$N(W', W) = \binom{W-1}{W'-1} < \binom{W}{W'}. \tag{5.26}$$

Combining these results we obtain,

$$\triangle_{\mathcal{R}^{W'}(\mathcal{F}_n^{\Phi})}(k) < \binom{W}{W'}\left(\frac{ek}{W'}\right)^{W'} \tag{5.27}$$

when $k \geq W'$. It is easily verified that,

$$\binom{W}{W'} < \frac{W^{W'}}{W'!} \tag{5.28}$$

and hence for $k \geq W'$,

$$\triangle_{\mathcal{R}^{W'}(\mathcal{F}_n^{\Phi})}(k) < \left(\frac{Wek}{W'}\right)^{W'}. \tag{5.29}$$

As in the proof of theorem 5.3, we let $\gamma = \frac{1}{2}$ and apply theorem 5.2. Inserting the upper bound of equation 5.29 and the proposed value for k_0 and re-arranging yields,

$$\mathcal{P} < 8\left(2e\frac{\epsilon}{32W}\ln\frac{32W}{\epsilon}\right)^{W'}. \tag{5.30}$$

In the proof of theorem 5.3 we obtained the inequality,

$$8\left(2e\frac{\epsilon}{32}\ln\frac{32}{\epsilon}\right)^{W'} < 8\exp(-1.5W'). \tag{5.31}$$

The final step is thus to prove that,

$$8\left(2e\frac{\epsilon}{32W}\ln\frac{32W}{\epsilon}\right)^{W'} \leq 8\left(2e\frac{\epsilon}{32}\ln\frac{32}{\epsilon}\right)^{W'} \tag{5.32}$$

when $W > W'$. This is true if,

$$\left(\frac{1}{W}\ln\frac{32W}{\epsilon}\right)^{W'} \leq \left(\ln\frac{32}{\epsilon}\right)^{W'} \tag{5.33}$$

for $0 < \epsilon \leq \frac{1}{4}$ and $W \geq 5$. This inequality can be proved by noting that,

$$\left(\frac{1}{W}\ln\frac{32W}{\epsilon}\right)^{W'} = \left(\frac{1}{W}\ln\frac{32}{\epsilon} + \frac{1}{W}\ln W\right)^{W'}. \tag{5.34}$$

We therefore need to show that for $W \geq 5$,

$$\frac{1}{W} \ln \frac{32W}{\epsilon} \leq \ln \frac{32}{\epsilon} \tag{5.35}$$

which is true if,

$$\frac{1}{W} \ln W \leq \frac{W-1}{W} \ln \frac{32}{\epsilon}. \tag{5.36}$$

Setting ϵ at its maximum allowable value of $\epsilon = \frac{1}{4}$ such that the term $\ln \frac{32}{\epsilon}$ takes its smallest possible value, we now need to show that,

$$\frac{1}{W} \ln W \leq \frac{W-1}{W} \ln 128 \leq \frac{W-1}{W} \ln \frac{32}{\epsilon} \tag{5.37}$$

or

$$\ln W \leq \ln(128^{W-1}) \tag{5.38}$$

which is clearly true for $W \geq 5$. This completes the proof of theorem 5.8. $\square$

As was the case for theorem 5.3 it is easily verified that, because theorem 5.2 and our upper bound on $\triangle_{\mathcal{F}_n^\Phi}(i)$ hold when the environment X is a subset of $\mathbb{R}^n$, theorem 5.8 also holds when this is the case.

What does this theorem tell us? Self-structuring is in general a means of finding the smallest Φ-net capable of learning to classify a set of training examples to an acceptable accuracy. The confidence value provided by this theorem is again very close to unity for realistic values of W', and if W' is small enough in comparison to W then we can obtain values of k_0 which are significantly smaller than those provided by theorem 5.3, which corresponds to the case where we allow as many as W weights to be non-zero. The theorem therefore suggests that by training using a self-structuring algorithm we might expect to require less training data than when using a standard training algorithm in order to obtain the same generalization performance.

In order to further illustrate this we now use the bounds derived in theorems 5.3 and 5.8 to make a comparison between the bounds on the sufficient numbers of training examples in the two different cases. Assume that on the one hand we have a Φ-net with W weights, trained using a standard training algorithm. We compare this to a Φ-net having W weights in total which we train using a self-structuring algorithm such that we attempt to find a solution with at most W' non-zero weights. In the former case, the sufficient number of training examples is,

$$k_0^{(1)} = \frac{32W}{\epsilon} \ln \frac{32}{\epsilon} \tag{5.39}$$

where $0 < \epsilon \leq \frac{1}{4}$, and this leads to a confidence of $1 - 8\exp(-1.5W)$. In the latter case the sufficient number of training examples is,

$$k_0^{(2)} = \frac{32W'}{\epsilon} \ln \frac{32W}{\epsilon} \tag{5.40}$$

where $0 < \epsilon \leq \frac{1}{4}$, leading to a confidence of $1 - 8\exp(-1.5W')$. In both cases ϵ is set to the same value, and we assume that the network learns the necessary fraction $(1 - \frac{\epsilon}{2})$ of the training examples[5]. Although the actual confidence values obtained are different, they are both extremely close to unity for any realistic value of W or W', and in fact differ by only a very small amount. We therefore disregard this difference and simply compare $k_0^{(1)}$ and $k_0^{(2)}$ for appropriate values of W, W' and ϵ. As an example we use $W = 500$ and examine values of W' ranging from 20 to 450, with four different values of ϵ. Figure 5.6 illustrates the resulting values for

[5]Because, in applying the above methods, the VC dimension for the case involving self-structuring can be expected to be lower than that for the case without self-structuring, this is again an important assumption. As in subsection 5.3.2, we note that its validity will be dependent on the specific characteristics of the network used, and on the actual problem addressed.

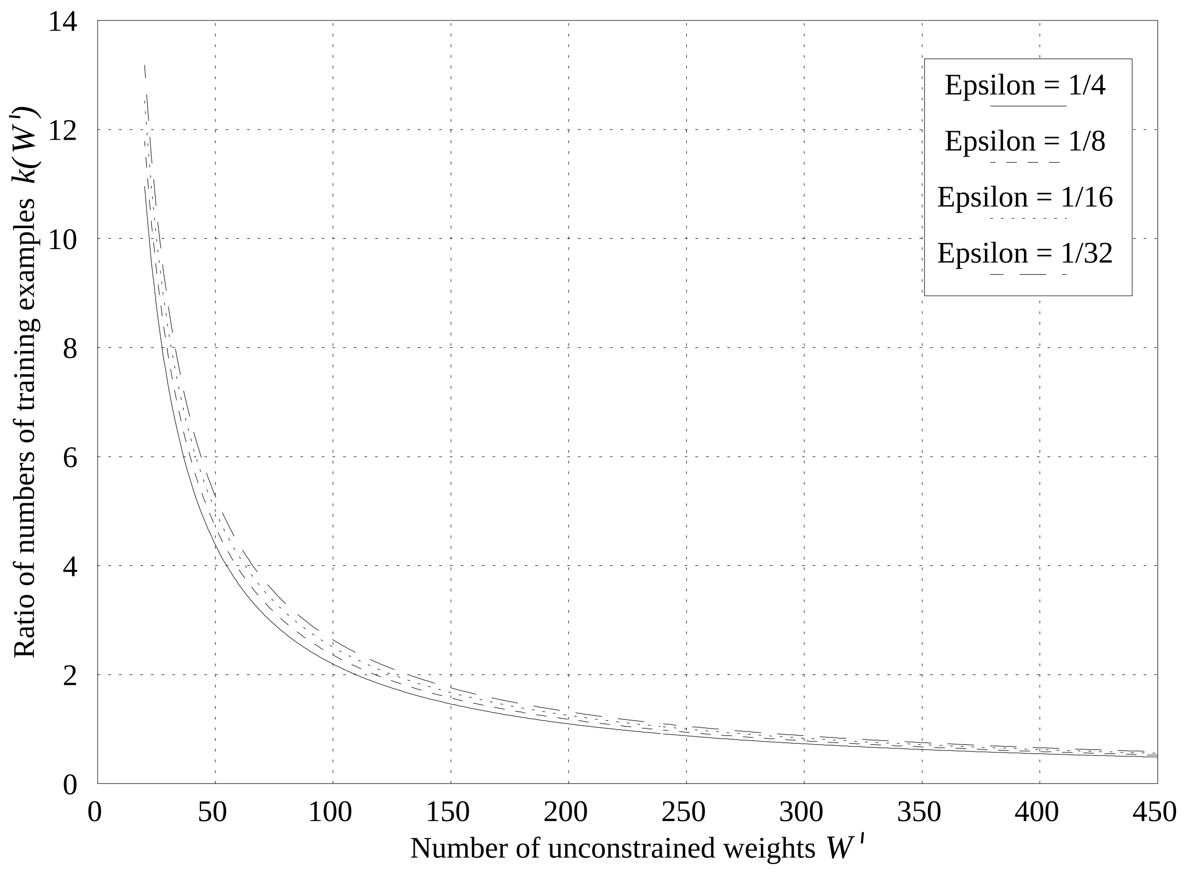


Figure 5.6: Comparison of sufficient numbers of training examples using the bounds derived for cases with and without self-structuring. For both cases the total number of weights is $W = 500$, and in the case with self-structuring W' denotes the number of unconstrained weights. The quantity $k(W')$ is defined in full in the text.

$k(W') = (k_0^{(1)}/k_0^{(2)})$. Clearly, a significant saving in the number of training examples appears to be possible, particularly if self-structuring significantly reduces the number of weights in the network.

Notice that for high values of W' we in fact obtain $(k_0^{(1)}/k_0^{(2)}) < 1$. This is surprising because we would like (and expect) the value of $k_0^{(2)}$ to approach that of $k_0^{(1)}$ as W' approaches W. The ratio is incorrect for large W' simply because the bound on the growth function used in proving theorem 5.8 (equation 5.29) is a significant overestimate when W' approaches W, and consequently the W in the logarithmic term of $k_0^{(2)}$ must be introduced to compensate (if it is not introduced then the inequality of equation 5.32 no longer holds). Note that this suggests that for relatively small values of W' we may be significantly *underestimating* the degree to which the sufficient number of training examples can be reduced.

In addition to telling us about the utility of self-structuring training algorithms, our observation is important in the light of the criticism of theorem 5.3 that we raised earlier: that the bounds it provides are impractical. Our new theorem allows us to reduce one of the upper bounds significantly; in the above example, which is reasonably realistic, the bound can be reduced by as much as a factor of about 10.

In [79], the application of self-structuring to feedforward networks of LTEs was considered. Specifically, networks having $N' \geq 2$ LTEs and $E' \geq N'$ edges where considered, and training algorithms were required to find networks having at most $E \geq 2$ edges with non-zero weights, and at most $N \geq 2$ nodes having non-zero weights on incoming edges. A sufficient number $k_0 = \frac{32W}{\epsilon} \ln \frac{32NE'}{\epsilon}$ of training examples was obtained with a corresponding confidence value of $1 - 8\exp(-1.5W)$, where $W = E + N$.

5.5 Discussion

5.5.1 The Interpretation of the Comparative Results

The results we have obtained in this chapter on necessary and sufficient numbers of training examples for Φ-nets are quite easily interpreted (subsection 5.2.3) and we will say no more about them here. Some comments are in order however on the manner in which the comparative results presented in sections 5.3 and 5.4 should be interpreted.

Our comparison between Φ-nets with fixed basis functions and feedforward networks of LTEs suggests that an approach in which we make a choice of fixed basis functions and then train the resulting Φ-net using a simple optimization algorithm for $\mathbf{w}$ may be preferable to the more usual approach of training a multilayer feedforward network having the same number of weights using a nonlinear optimization technique such as the HLBP algorithm. This is because in the former case training is invariably significantly faster, and because we obtain the observed reduction in the sufficient number of training examples for valid generalization. As we pointed out earlier, we have to assume that each network can learn a sufficient number of training examples to begin with in order for our comparison to be valid.

This is precisely the conclusion that we would expect, given the usual intuitive understanding of the relation of generalization ability to capacity; what we have actually done is to formalize the intuition that given the choice between two alternative networks capable of learning a set of training examples, the one with the least capacity will in general provide the better generalization (provided, of course, that they are both capable of computing a function close to the 'best' one). Given the results of chapter 4, Φ-nets with fixed basis functions can be expected to have lower capacity (measured using the VC dimension) in general. Note however that hybrid training algorithms appear to allow us to exploit the full capacity available with adapting basis functions in learning the training examples, but to apply the smaller capacity associated with fixed basis functions for the purposes of considering generalization. A similar intuitive explanation, in terms of capacity, can be applied to the results of section 5.4 relating to self-structuring.

For the same reasons as mentioned in chapter 4, the situation is likely to move further in favour of Φ-nets if we use linear sigmoid functions instead of LTEs. Furthermore, we do not think that improvements to the results we have used from [79] will necessarily change our conclusions, because we have used exactly analogous proof techniques and hence our own results should immediately be subject to improvement in the same way.

Finally, it is important to appreciate that our results in no way imply that Φ-nets are superior to feedforward networks for all problems. The results are based on very general, worst case bounds, and consequently should be interpreted as *general* results about the behaviour of different networks rather than predictions regarding behaviour in specific cases. Again, similar comments apply to our results on self-structuring.

5.5.2 Improving the Main Results

There is some potential for improving the bounds that we have derived for Φ-nets with fixed basis functions, in particular those providing sufficient conditions. In deriving these results we specifically aimed to obtain bounds which would allow us to make comparisons with the existing results for feedforward networks of LTEs, and so we tried to produce results which were directly comparable to those of [79]. The results for both of the relevant types of network, including those derived in order to consider self-structuring, are all based on theorem 5.2. The constant in this theorem has recently been improved from 8 to 4 by Anthony and Shawe-Taylor [74], and so there is immediate potential for improving the relevant bounds. Unfortunately, the improvement is still unlikely to make the bounds fully practical. We suspect that further improvements to theorem 5.2 are possible.

The bounds derived in this chapter relating to sufficient conditions may also be open to further improvement if we use the tight bounds on the growth function for Φ-nets derived in lemma 4.4, rather than the bounds based on Sauer's lemma employed in proving theorems 5.3 and 5.8.

Our bounds relating to necessary conditions are at present quite stable, because the bounds on VC dimensions for Φ-nets that we derived in chapter 4 are tight[6] and because we know of no existing improvements to theorem 5.4.

5.5.3 Other Bounds

By using the VC dimension results derived in chapter 4 it is possible to obtain further bounds on sufficient numbers of training examples for Φ-nets with fixed basis functions, trained using standard training algorithms. In order to do this we use existing results obtained from theorems similar to theorem 5.2. For example, the following result is given in [87].

[6]Again, the bounds on the VC dimensions for radial basis function networks with adapting centres are a possible exception here, and improvements to these bounds would allow us to obtain improved necessary conditions for this case.

Theorem 5.9 *Let $0 < \epsilon$, $\delta < 1$ and $0 < \gamma \leq 1$ and consider a probability distribution P' on $X \times \{0,1\}$. If the class $\mathcal{F}$ of functions $f : X \to \{0,1\}$ has finite VC dimension d then for,*

$$k > k_0 = \frac{1}{\gamma^2 \epsilon (1 - \sqrt{\epsilon})} \left[4 \ln \left(\frac{4}{\delta} \right) + 6d \ln \left(\frac{4}{\gamma^{2/3} \epsilon} \right) \right] \tag{5.41}$$

the probability that a network computing the class of functions $\mathcal{F}$ can misclassify at most a fraction $(1-\gamma)\epsilon$ of a sequence T_k of k training examples and have error $\mathrm{er}_{P'}(f) \leq \epsilon$, where $\mathrm{er}_{P'}(f)$ is as defined in chapter 3, equation 3.15, is at least $1 - \delta$.

Recall the simple example that we introduced in section 5.3 for a Φ-net having $W = 200$ weights. Under the reasonable assumption (see the discussion in chapter 4) that in this case $\mathcal{V}(\mathcal{F}) = 200$, using the same values for ϵ and δ of $\epsilon = \frac{1}{4}$ and $\delta \simeq 4 \times 10^{-130}$, and the appropriate value for γ of $\gamma = \frac{1}{2}$, this result provides a value of,

$$k_0 \simeq 162,500. \tag{5.42}$$

This is worse than the value obtained previously using our result. Furthermore, theorem 5.9 was derived using the improved version of theorem 5.2 mentioned above, which suggests that our result can indeed be improved further as suggested. Theorem 5.9 does have the advantage that it allows us to obtain bounds for larger values of δ than we can deal with using theorem 5.3.

A further result is presented in [73] which can also be used to obtain further bounds, and similar comments apply. Also, techniques are introduced in [87] which may allow us to extend our results to Φ-nets with several outputs.

5.5.4 The Need for Experimental Studies

The theory presented in this chapter has allowed us to make various suggestions regarding the use of connectionist networks in practice; in particular, we have made suggestions regarding the use of hybrid training algorithms, the use of Φ-nets versus multilayer feedforward networks, and the use of self-structuring training algorithms. It is important that these suggestions should now be investigated experimentally in order to further assess them. We have not attempted such an investigation, as we believe that in order to obtain significant results, quite an involved and long-term study will be required, and this falls outside the scope of the present work.

5.5.5 Other Formalisms

We end this section with some brief comments on the use of alternative formalisms (see chapter 3) for analysing the generalization performance of Φ-nets.

We have already pointed out that although the sufficient conditions derived herein provide very powerful and general results, the resulting bounds on numbers of training examples are rather impractical. This is a well-known problem with bounds derived using this type of computational learning theory, and has also been noted by other authors in other circumstances. An experimental study which investigates the quality of this type of bound has been performed by Cohn and Tesauro [132], who investigate the way in which the average generalization performance of LDFs and multilayer feedforward networks, applied to various different problems, compares with worst case VC dimension type bounds. They find that, as expected, the average generalization obtained can be significantly better than that suggested by these bounds. However, they also consider a distribution independent upper bound derived by Haussler *et al.* [92] which applies to a Bayes-optimal learning algorithm, and find that this performs significantly better. A detailed consideration of Φ-nets using the methods of [92] would therefore be interesting. Similarly, further consideration of Φ-nets using the techniques based on statistical physics mentioned in chapter 3 would be useful.

In chapter 3 we also mentioned the theoretical framework for generalization introduced by Wolpert [99]. This work considers Φ-nets as a special case. Specifically, it considers Φ-nets with real-valued outputs which compute functions $\sum_{i=1}^{m} w_i \phi_i(\mathbf{x})$ where $w_m \neq 0$, the functions ϕ_i are infinitely differentiable and non-zero and m is as small as possible. It is assumed that a network learns the available training examples exactly. Wolpert proves that PDFs are the only Φ-nets of this type that are able to obey a particular invariance requirement forming part of the formalism. The consequence of this is that the use of any other set of basis functions implies the introduction of a preferred origin, scaling dimension, orientation or combination of the three. Thus, if these preferences are inconsistent with the problem being addressed then a PDF is the preferred Φ-net in the context of generalization subject to the conditions given above.

5.6 Conclusion

In this chapter we have used various techniques from PAC learning theory to analyse the ability of Φ-nets to generalize, concentrating on the number of training examples that must be learnt if we are to be confident that the network will provide a specified performance when used to classify further examples. We have derived various new bounds which provide necessary and sufficient conditions on the required number of training examples for both general and specific Φ-nets, and we have extended one of our sufficient conditions in an attempt to take into account the effect of using a self-structuring training algorithm. We have also used our results, along with an intuitive argument, to attempt to justify the use of hybrid training algorithms. Our results apply regardless of the probability distribution governing the occurrence of examples, and our most general results apply to all Φ-nets, including all the standard networks described in chapter 2.

Some of the results obtained in this chapter provide an extension of the work of Baum and Haussler [79] to a new class of networks; the results in [79] apply to a class of networks similar to multilayer perceptrons, and we have compared the bounds for these networks with the new bounds obtained for Φ-nets. The results of this comparison suggest that under certain circumstances Φ-nets may be preferable to feedforward networks of LTEs and multilayer perceptrons in terms of their ability to generalize. We have also performed a similar comparison using our results on self-structuring; this comparison suggests that in a similar manner it may be advantageous to use self-structuring training algorithms rather than standard training algorithms.

Part III

Self-Structuring

Chapter 6

Self-Structuring Training Algorithms for Φ-Nets

6.1 Introduction

In this chapter we address the second of the main aims of this dissertation: the theoretical analysis of the manner in which the best architecture for a Φ-net can be selected, as well as the values of the corresponding weights. On the basis of our analysis, we then derive a new self-structuring training algorithm for Φ-nets, and we test its performance experimentally.

The underlying reasons for studying self-structuring were mentioned briefly in chapter 1, and we expand on them below. Self-structuring has been studied by various authors (see section 6.3), most of whom consider multilayer perceptrons, and most of whom use rather ad hoc techniques; furthermore, there has tended to be little or no attempt to determine whether the network architectures obtained are the optimum (in an appropriate sense), or even close to optimum, architectures, and some of the techniques that have been used are highly computationally demanding. We saw in chapter 2 that Φ-nets with fixed basis functions can be considered, for the purposes of training, to be essentially linear systems, and that training can consequently be re-cast as a linear least squares problem. We will exploit this fact in the following work in order to provide a rigorous analysis of the problem; this analysis leads directly to our self-structuring training algorithm.

One of the contributions made in this chapter is of particular importance. Perhaps the most common technique for performing self-structuring presented in the literature, which is commonly known as the *weight decay* technique, involves starting with a relatively large network and attempting to remove weights by setting them to zero using a technique that attempts to minimize the Euclidean norm of a vector containing all the weights. There is an obvious assumption involved here, namely that the weight vector having the minimum Euclidean norm among the set of all weight vectors that perform satisfactorily on the training examples is the weight vector in that set having the largest number of zero elements. We show that in certain, realistic circumstances this assumption is wrong for Φ-nets with fixed basis functions, and we argue intuitively that it is likely to be incorrect for other types of network. We also suggest a simple modification to the weight decay technique on the basis of our main analysis. Our main result in fact provides a definite lower bound on the extent to which self-structuring is possible in specific cases for this type of connectionist network; to our knowledge this is the only result of this type obtained to date.

In section 6.2 we discuss the way in which the 'best' architecture for a Φ-net with fixed basis functions should be defined. In section 6.3 we outline the various different approaches to self-structuring that are available and review some recent results on the computational complexity of this type of problem. In section 6.4 we describe and motivate the use of the particular approach to self-structuring studied in the chapter, and in section 6.5 we provide the main analysis of this approach to self-structuring; this analysis leads us to suggest a modification to the well-known weight decay training technique. In section 6.6 we use the preceding analysis to derive a self-structuring training algorithm which we then test experimentally. In section 6.7 we extend our initial analysis of the self-structuring problem. In section 6.8 we suggest ways in which our approach can be improved and we discuss some alternative approaches; we also briefly discuss our suggested modification to the weight decay technique. Section 6.9 concludes the chapter.

Some of the research presented in this chapter has been published in Lynch *et al.* [133, 134], although once again the exposition provided here is significantly expanded. In order to maintain mathematical tractability we assume in this chapter that $\mathcal{C} = \mathbb{R}$, although the use of $\mathcal{C} = \mathbb{B}$ is discussed briefly in section 6.8.

6.1.1 Notation

The notation $\mathbb{R}^{i \times j}$ is used to denote the set of all i-by-j matrices with real-valued elements. Also, for any matrix $\mathbf{A} \in \mathbb{R}^{i \times j}$ we let rank($\mathbf{A}$) denote the rank of the matrix. The Euclidean norm of a vector $\mathbf{a} \in \mathbb{R}^i$ is, as usual, denoted by $\|\mathbf{a}\|$. We use the notation $\mathbf{0}_{i \times j}$ and $\mathbf{I}_i$ to denote the i-by-j matrix of zeros and the i-by-i identity matrix respectively; the dimensions are omitted when they are clear from the context. We use the notation diag$(a_1, a_2, \ldots, a_i)$ where each a_j is real to denote the diagonal matrix,

$$\mathbf{A} = \begin{bmatrix} a_1 & & & \\ & a_2 & & \\ & & \ddots & \\ & & & a_i \end{bmatrix} \tag{6.1}$$

where all elements not on the main diagonal are zero. The notation span$\{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_i\}$ denotes the set of all linear combinations of the vectors $\mathbf{a}_j$ where $j = 1, 2, \ldots, i$, that is, the set of all vectors of the form,

$$\lambda_1 \mathbf{a}_1 + \lambda_2 \mathbf{a}_2 + \cdots + \lambda_i \mathbf{a}_i, \tag{6.2}$$

where $[\ \lambda_1\ \ \lambda_2\ \ \cdots\ \ \lambda_i\ ]^T \in \mathbb{R}^i$. The *row space* rs($\mathbf{A}$) of a matrix $\mathbf{A}$ is the set of all linear combinations of its rows and the *column space* cs($\mathbf{A}$) is the set of all linear combinations of its columns. Finally, we use the notation dim(V) to denote the dimension of a vector space V.

6.2 How do we Define the 'Best' Architecture?

In chapter 1 we argued briefly that in general it is desirable to minimize the size of a network because this reduces storage requirements and because it reduces the amount of computation required, both to classify an input and to determine the required weight values during training (the latter case, of course, applies most directly when the network architecture is fixed; when the architecture is variable the situation is more complicated). It was also stated in chapter 1, and argued briefly in chapter 4, that minimizing the size of a network is in general advantageous in terms of generalization performance; the theoretical results presented in the last chapter suggest that this is true in the case of Φ-nets with fixed basis functions, and that the number W of variable weights is an appropriate measure of size in this case.

It is more difficult to define a single, appropriate measure of the size of a Φ-net for the purposes of considering storage and computation requirements, because a truly optimum measure will depend on the precise nature of the basis functions that are available for the purposes of constructing a network. For example, storage requirements are obviously lower for a PDF basis function such as $\phi_i(\mathbf{x}) = x_1^2$ than for a RBFN type basis function, because in the former case we do not have to store a centre $\mathbf{y}_i \in \mathbb{R}^n$. In order to maintain a degree of generality we will use W as a measure of size; it is clearly an appropriate approximate measure.

Consequently, when we refer to the 'size' of a Φ-net in this chapter, we assume that size is measured as the number W of weights in $\mathbf{w}$. Our approach to self-structuring will involve trying to find the smallest network capable of providing a specified performance on the available training examples, and is defined fully in section 6.4 below. Note that although approaches which minimize size can also be applied to types of network other than Φ-nets, different measures of size may be appropriate.

6.3 Basic Approaches to Self-Structuring

An obvious method of searching for an optimum architecture for a specific problem is to train, using a standard training algorithm, a large collection of networks with different architectures and see which performs best. In the present scenario we would search for the smallest network capable of providing an acceptable error on the training examples. This is clearly not a good approach because it represents an unrealistic computational effort, even for Φ-nets with fixed basis functions, for which training can often be accomplished relatively quickly. The reason for this is that the total number N of architectures which can be tested will generally be very large. For example, for a Φ-net derived from a fixed set Φ of m fixed basis functions we have,

$$N(m) = \sum_{i=1}^{m+1} \binom{m+1}{i}, \tag{6.3}$$

where a single architecture corresponds to a Φ-net constructed using some choice of m' basis functions from those available where $0 \leq m' \leq m$. The value $N(m)$ becomes very large very quickly; for example, we have $N(10) = 2047$, $N(20) = 2097151$, and $N(50) \simeq 2 \times 10^{15}$. For networks such as MLPs the situation is likely to be even worse.

A much better approach is to develop a training algorithm which attempts to determine an optimum architecture using the available training examples. We might however expect that, in the light of equation 6.3, the task of finding a true optimum solution, that is, the smallest possible network, is computationally very difficult. Although no extensive study has been made of the computational complexity of architecture selection, this proposition is supported by two results due to Lin and Vitter [135]. These authors have considered the design of networks that compute linear threshold functions which are the same as those of definition 4.1 but for the fact that they produce outputs in the set $\{0, 1\}$ rather than $\mathbb{B}$. They also consider the design of feedforward networks constructed from these linear threshold functions but otherwise as described in our definition of a feedforward network (definition 4.17). Consider the following problem, called the *optimal consistent network problem.*

Definition 6.1 (Optimal consistent network problem)

Instance: *A set of training examples and a positive integer i. Inputs to the network are either in* $\{0, 1\}^n$ *or* $\mathbb{R}^n$*.*

Question: *Is there a feedforward network which is consistent with the training examples and such that the size S of the network, defined as the number of computation nodes, satisfies $S \leq i$?*

Lin and Vitter show that this problem is **NP**-complete. They also consider the following problem, called the *optimal consistent perceptron problem*, which is of greater relevance to our investigation.

Definition 6.2 (Optimal consistent perceptron problem)

Instance: *A set of training examples, inputs of which are in $\{0,1\}^n$.*

Problem: *Construct a linear threshold function which is consistent with the training examples and such that the number of non-zero weights is minimized.*

Lin and Vitter show that this problem is **NP**-hard.

There are three obvious ways in which we can approach the problem of architecture selection in practice (which we mentioned briefly in chapter 4). Firstly, we can start with the smallest possible architecture and increase its complexity until a network capable of an acceptable level of performance is obtained. We hope that this network has a size near to the smallest possible. This is the strategy used by the GMDH algorithm [9], and the approach has also been used by, for example, Debenham [136] and Kadirkamanathan [137] for RBFNs with adapting basis functions, and by Fahlman and Lebiere [131] for MLP type networks.

Secondly, we can begin with a large network — that is, one which is assumed or known to be large enough to solve the problem in question — and remove weights or nodes in order to obtain a smaller network, which we again hope has a size near to the smallest possible. This approach has been used for MLP type networks by Le Cun *et al.* [138], and is also the approach used in the research presented in this chapter for reasons which we give below.

Finally, we can use a combination of the previous two approaches. This strategy has been used by Alpaydin [139].

As usual, the literature for this area of study is rather large and consequently we will not attempt an exhaustive review; the references cited above represent a subset of the literature available. A more extensive review can be found in [139] (see also Nelson and Rogers [140]).

6.4 The Self-Structuring Problem for Φ-Nets

In order to analyse self-structuring in this chapter we use an approach which is most similar in spirit to the second method suggested above: we start with a large network and try to find a smaller one which provides comparable performance. Specifically, we address the following problem.

Definition 6.3 (The self-structuring problem) *Assume we have a Φ-net,*

$$\mathcal{N} = (n, m, \Phi, \mathbf{w}, \mathbb{R}) \tag{6.4}$$

which we use to solve a problem represented by a sequence T_k of k training examples. Let the error of $\mathcal{N}$ on the training examples (which we define fully below) be ξ_{opt}. What is the smallest network,

$$\mathcal{N}' = (n, m', \Phi', \mathbf{w}', \mathbb{R}) \tag{6.5}$$

where $\Phi' \subseteq \Phi$ which is also capable of providing an error of ξ_{opt}?

The reason for addressing self-structuring in this manner is that it allows us to obtain a good theoretical hold on the problem. Note that an underlying assumption here is that we begin with a network $\mathcal{N}$ which is thought or known to be large enough to achieve an acceptable error ξ_{opt}. We then attempt to reduce the size of $\mathcal{N}$ for the reasons stated above. Furthermore, note that by beginning with a set Φ containing a wide variety of basis functions, perhaps corresponding to various different standard Φ-nets as described in chapter 2 (and perhaps much larger than we expect is necessary), we can regard the self-structuring process described in definition 6.3 as a method of choosing basis functions that are well suited to the problem being addressed.

Note that, in our original definition of a Φ-net, the unit term in the extended vector, which corresponds to the weight w_0, is regarded as being separate from the basis functions. Clearly we can alternatively regard this term as a basis function which has the value 1 for all inputs and include it in Φ while removing the unit term from the extended vector. This provides an exactly equivalent Φ-net, although the *notational* alteration is required for the purpose of interpreting definition 6.3 correctly.

Although this way of considering self-structuring is most similar to one in particular of the approaches outlined above, it is also applicable in some circumstances when considering the others. This is because, in some cases, even when we allow the size of a network to start small and increase during training, the set Φ from which we can choose basis functions to add may be fixed in advance; for example, Φ could be the set of all PDF basis functions for PDFs of some fixed degree d.

6.5 Which Network is the Smallest?

We now provide an analysis of the self-structuring problem defined in definition 6.3. We begin by reviewing the way in which the task of training the network $\mathcal{N}$ using a standard training algorithm (one which does not attempt to optimize the architecture) can be cast as a least squares problem. We then address the problem of how to search for a network $\mathcal{N}'$.

At various points in this section we introduce and briefly review some standard least squares techniques. A complete account of these techniques can be found in Press *et al.* [26], Haykin [27], Ljung [141] and Golub and Van Loan [142].

6.5.1 Training as a Least Squares Problem

We begin by reviewing the way in which the task of training a Φ-net with fixed basis functions can be regarded as a least squares problem. Assume, as usual, that we have a sequence $T_k = ((\mathbf{x}_1, o_1), (\mathbf{x}_2, o_2), \ldots, (\mathbf{x}_k, o_k))$ of k training examples where $\mathbf{x}_i \in \mathbb{R}^n$ and $o_i \in \mathbb{R}$ for $i = 1, 2, \ldots, k$. Also, we have an initial Φ-net $\mathcal{N} = (n, m, \Phi, \mathbf{w}, \mathbb{R})$ as introduced in definition 6.3 and defined in chapter 2, and so we can form extended vectors $\tilde{\mathbf{x}}_i = \boldsymbol{\varphi}(\mathbf{x}_i)$ for $i = 1, 2, \ldots, k$. We define $\mathbf{P} \in \mathbb{R}^{k \times (m+1)}$ to be the matrix containing as rows the extended training inputs,

$$\mathbf{P} = \begin{bmatrix} \boldsymbol{\varphi}^T(\mathbf{x}_1) \\ \boldsymbol{\varphi}^T(\mathbf{x}_2) \\ \vdots \\ \boldsymbol{\varphi}^T(\mathbf{x}_k) \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{x}}_1^T \\ \tilde{\mathbf{x}}_2^T \\ \vdots \\ \tilde{\mathbf{x}}_k^T \end{bmatrix} \tag{6.6}$$

and we define $\mathbf{o} \in \mathbb{R}^k$ to be the vector containing the training outputs (which we will also refer to as the *desired* outputs),

$$\mathbf{o}^T = [\ o_1 \quad o_2 \quad \cdots \quad o_k\ ]. \tag{6.7}$$

The output of the Φ-net in response to some input vector $\mathbf{x} \in \mathbb{R}^n$ is $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \varphi(\mathbf{x}) = \mathbf{w}^T \tilde{\mathbf{x}}$. We define the error $\epsilon_i(\mathbf{w})$ for the ith training example as,

$$\epsilon_i(\mathbf{w}) = o_i - f_{\mathbf{w}}(\mathbf{x}_i). \tag{6.8}$$

As usual, we model the process of training the network as the process of searching for a weight vector $\mathbf{w}$ which minimizes the error $\xi(\mathbf{w})$, defined as,

$$\begin{aligned} \xi(\mathbf{w}) &= \sum_{i=1}^{k} \epsilon_i^2(\mathbf{w}) \\ &= \sum_{i=1}^{k} [o_i - f_{\mathbf{w}}(\mathbf{x}_i)]^2 \\ &= \sum_{i=1}^{k} [o_i - \mathbf{w}^T \tilde{\mathbf{x}}_i]^2, \end{aligned} \tag{6.9}$$

for the particular training sequence T_k. In order to determine the values for $\mathbf{w}$ which minimize $\xi(\mathbf{w})$ we differentiate equation 6.9 with respect to $\mathbf{w}$ and equate the result to zero. This leads to the well-known *normal equations*,

$$\mathbf{R}\mathbf{w} = \mathbf{p} \tag{6.10}$$

where,

$$\mathbf{R} = \sum_{i=1}^{k} \tilde{\mathbf{x}}_i \tilde{\mathbf{x}}_i^T = \mathbf{P}^T \mathbf{P} \tag{6.11}$$

and

$$\mathbf{p} = \sum_{i=1}^{k} \tilde{\mathbf{x}}_i o_i = \mathbf{P}^T \mathbf{o}. \tag{6.12}$$

Direct solution of the normal equations is not in general the best way to calculate $\mathbf{w}$ numerically [26], however for the purposes of our further development below the normal equations provide the most convenient starting point.

We now make an important observation. It is tempting at this point simply to assume that the matrix $\mathbf{R}$ is non-singular. If this is the case then there is a unique weight vector $\mathbf{w}_{\text{opt}} = \mathbf{R}^{-1}\mathbf{p}$ which provides an optimum error $\xi_{\text{opt}} = \xi(\mathbf{w}_{\text{opt}})$, and consequently we cannot perform self-structuring without increasing the error. Furthermore, although we can obviously remove any basis function corresponding to a zero weight in $\mathbf{w}_{\text{opt}}$, we can only regard the resulting reduction in the size of the network as a lucky outcome; it is not self-structuring of the type in which we are interested. However, although the assumption that $\mathbf{R}$ is non-singular is often correct in other applications of least squares techniques, it is not necessarily correct when considering the training of connectionist networks[1]. This observation is important, and we now elaborate on it. We will see below that when $\mathbf{R}$ is singular we have significantly more flexibility in reducing the size of the network $\mathcal{N}$ without increasing $\xi(\mathbf{w})$. Of course, the case in which $\mathbf{R}$ is non-singular is still relevant, and we extend our analysis to take this case (among others) into account in section 6.7; our extended analysis shows that when $\mathbf{R}$ is non-singular it may be possible to reduce the size of the network while causing only a small increase in $\xi(\mathbf{w})$.

6.5.2 The Matrix R can be Rank Deficient

We now show that, when training a network $\mathcal{N}$, the matrix $\mathbf{R}$ defined in equation 6.11 can be rank deficient, that is, we have $\text{rank}(\mathbf{R}) = r < m + 1$. Of course, when $\mathbf{R}$ is rank deficient, it is

[1]This was originally pointed out to the author by Dr. P. Rayner and Dr. M. Lynch.

also singular. The fact that $\mathbf{R}$ can be rank deficient in realistic circumstances has not previously been widely appreciated.

There are two distinct situations which can lead to rank deficiency when training a Φ-net $\mathcal{N}$ using the least squares techniques described, and we deal with them separately.

Rank Deficiency and the Set Φ

Recall that, in defining our version of the self-structuring problem, we suggested that the initial network $\mathcal{N}$ can be constructed by including many different types of basis function in Φ, and that in this case self-structuring can be regarded as a method for finding particularly suitable basis functions. If we use this approach then it is likely to be quite easy in practice to construct a network $\mathcal{N}$ for which $W > k$, where W and k are, as usual, the number of variable weights and the number of training examples respectively. Now, note that in this case we have,

$$\operatorname{rank}(\mathbf{R}) = \operatorname{rank}\left(\sum_{i=1}^{k} \tilde{\mathbf{x}}_i \tilde{\mathbf{x}}_i^T\right) \leq k < W = m + 1 \tag{6.13}$$

and consequently $\mathbf{R}$ is rank deficient. We will see below that as a consequence of this it is always possible to reduce the size of the network such that we obtain a new network $\mathcal{N}'$ for which $W \leq k$.

The result given in equation 6.13 may not seem obvious at first glance. To see that it is true, consider any square matrix $\mathbf{A} \in \mathbb{R}^{i \times i}$ having rank $r < i$; this means that the column space of $\mathbf{A}$ has dimension r. It is easily verified that the column space of $(\mathbf{A} + \mathbf{x}\mathbf{x}^T)$ for any vector $\mathbf{x} \in \mathbb{R}^i$ consists of vectors of the form,

$$\theta_1 \mathbf{a}_1 + \theta_2 \mathbf{a}_2 + \cdots + \theta_i \mathbf{a}_i + \theta_{i+1} \mathbf{x} \tag{6.14}$$

where the θ_j are reals and the $\mathbf{a}_j$ are the columns of $\mathbf{A}$, and consequently the matrix $(\mathbf{A} + \mathbf{x}\mathbf{x}^T)$ has a rank of either r or $(r + 1)$.

Some further justification is required here. The reader may wish to raise the following objection: it is well-known that when training a network we should in general use significantly more training examples than there are variable weights if we wish to obtain good performance after training. If we begin with more variable weights than training examples, are we merely using a network which will be likely to learn the training examples well, but will also be likely to suffer from the usual problems due to overfitting that are associated with this situation? Although this objection would be entirely fair if our aim was merely to train the initial network $\mathcal{N}$ using ordinary least squares techniques, while making no attempt at self-structuring, it is important to remember that this is not in fact the case. In the present scenario we are attempting to find a new network $\mathcal{N}'$ which has a reduced size. We expect that the final network $\mathcal{N}'$ is rather less likely to overfit the examples if, firstly, it has significantly fewer than k variable weights, and secondly, it is re-trained on the available examples, this time using a standard training technique such as ordinary least squares. In this context, self-structuring can be viewed as a technique for reducing or eliminating any overfitting which might occur as a result of using a large initial network. Indeed, this is one of the reasons often advanced for the use of self-structuring training algorithms.

Rank Deficiency and Specific Problems

We now demonstrate that the matrix $\mathbf{R}$ can be rank deficient even when the number of training examples used is significantly greater than the number of variable weights. We will do this

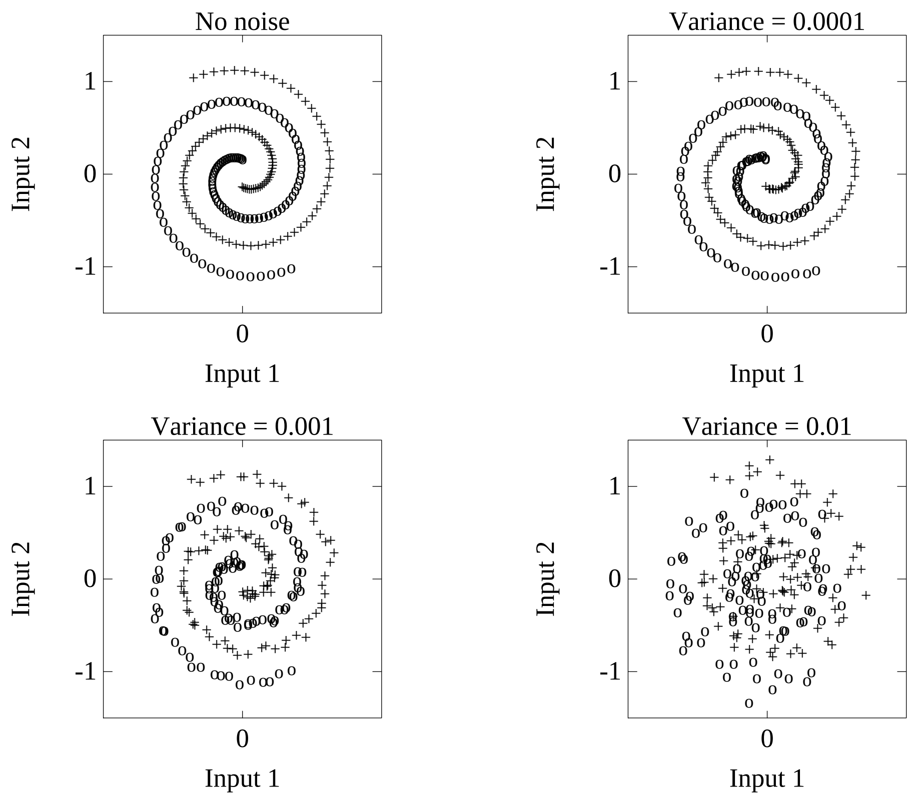


Figure 6.1: Four different instances of the two-spirals problem. Each instance has 100 examples in each class; members of class 1 are shown as a 'o' and members of class two as a '+'. The top left instance is not corrupted by noise; in the remaining instances the inputs are corrupted by additive zero mean Gaussian noise of the variances shown. In each of these diagrams the scale for input 1 is the same as that for input 2.

by examining the matrix $\mathbf{R}$ obtained when applying a particular Φ-net to a well-known two-class classification problem: the *two-spirals problem*. The particular Φ-net used is an (n, d) discriminator, as defined in chapter 2, which computes polynomial discriminant functions. We use this type of network because it is the Φ-net most often criticized for requiring too many weights, and hence the network which is likely to benefit the most from self-structuring.

The two-spirals problem has two inputs and two classes. The two classes are in the form of interlocking spirals; this is illustrated in figure 6.1 in which four different instances of the problem are shown. In each case there are 100 examples in each class. In one instance no noise has been added to the examples, and in the other three instances the inputs have been corrupted by additive zero mean Gaussian noise of the variances shown. This problem was originally introduced because it is an extremely difficult problem for the hidden layer back-propagation training algorithm to solve [131].

Figure 6.2 shows the way in which the rank of the matrix $\mathbf{R}$ for a $(2, 8)$ discriminator, that is, one which computes polynomial discriminant functions up to eighth degree, increases with the number of training examples for four instances of the two-spirals problem as illustrated in figure 6.1. In this case, the size of $\mathbf{R}$ is 45-by-45 (equation 2.25). Note that when the examples are not corrupted by noise we need to use nearly four times as many training examples as there

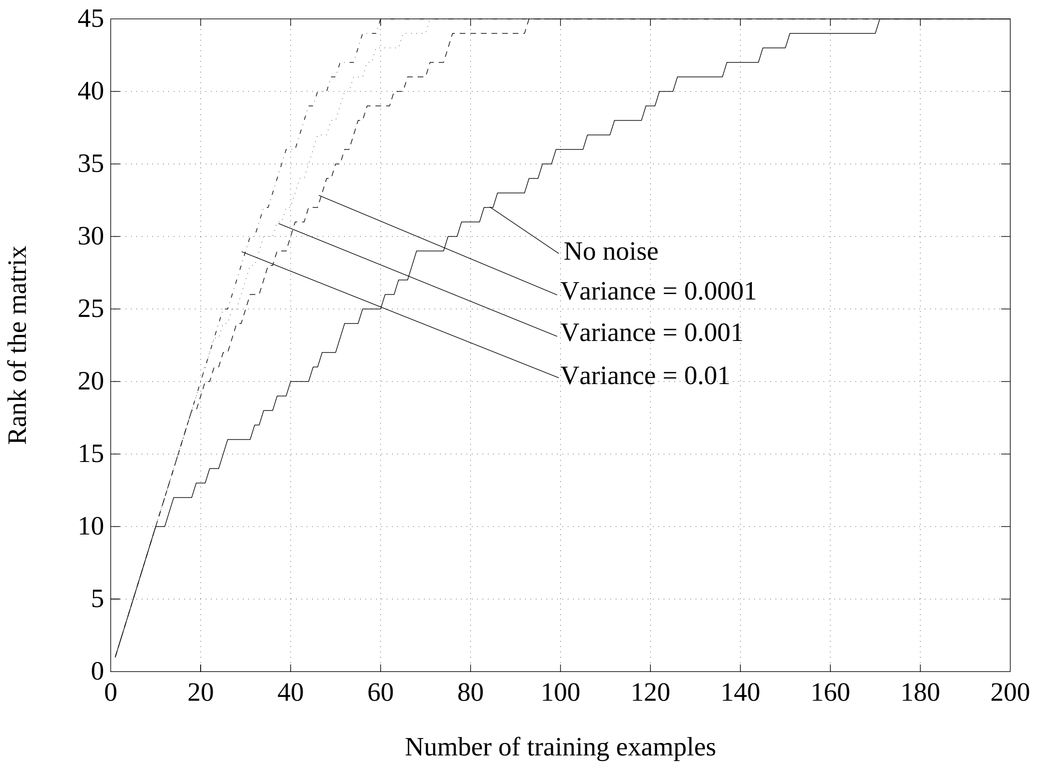


Figure 6.2: Increase in the rank of the matrix **R** with increasing number of training examples for four instances of the two-spirals problem as illustrated in figure 6.1.

are variable weights in order to make $\mathbf{R}$ non-singular. For a noise variance of 0.0001 this reduces to about twice as many examples as weights, and for variances of 0.001 and 0.01 the number required is less again, although still greater than the number of variable weights. In section 6.7 we extend our analysis in a manner which allows us to deal with any increase in rank caused by the presence of noise.

6.5.3 The Solution Space and the Minimum Norm Solution

The Solution Space

Having established that $\mathbf{R}$ can be rank deficient, and that this in fact needs to be the case if we are to have any genuine flexibility to reduce the size of $\mathcal{N}$ without increasing $\xi(\mathbf{w})$, we must now address the question of how the normal equations can be solved when $\mathbf{R}$ is singular. Throughout the remainder of this section we assume that $\text{rank}(\mathbf{R}) = r < m + 1 = W$.

We begin by defining the *range* of a matrix as follows.

Definition 6.4 *The* range $R(\mathbf{A})$ *of a matrix* $\mathbf{A} \in \mathbb{R}^{i \times j}$ *is the set of vectors* $\mathbf{x}$ *defined as follows.*

$$R(\mathbf{A}) = \{\mathbf{x} \mid \mathbf{x} = \mathbf{A}\mathbf{b} \text{ where } \mathbf{b} \in \mathbb{R}^j\}. \tag{6.15}$$

Clearly $R(\mathbf{A}) = \text{cs}(\mathbf{A})$.

Thus, if we consider $\mathbf{A}$ as a mapping, $R(\mathbf{A})$ can be interpreted as the range of the mapping; it is the set of all vectors that can be 'reached' by $\mathbf{A}$. It is a well-known result (see for example Green [117]) that if $\text{rank}(\mathbf{A}) = r$ then $R(\mathbf{A})$ is a subspace of $\mathbb{R}^i$ and $\dim(R(\mathbf{A})) = r$.

Given a completely general set of linear equations, say $\mathbf{A}\mathbf{b} = \mathbf{c}$, then the fact that $\mathbf{A}$ is rank deficient means that either the set of equations has no solution, or that it has an infinite number of solutions which constitute what we will refer to as the *solution space*. The former is the case if $\mathbf{c} \notin R(\mathbf{A})$ and the latter if $\mathbf{c} \in R(\mathbf{A})$. We now make the important observation that, because of their structure, a set of normal equations *always* has at least one solution. Although this result is intuitively appealing given the nature of the function $\xi(\mathbf{w})$, and is often hinted at in the literature, we have been unable to find a convincing general proof; as the result is by no means trivial to prove we now include a full derivation.

In order to obtain the required result we show that it is always the case that $\mathbf{p} \in R(\mathbf{R})$. Note that, because $\mathbf{p}$ is a linear combination of training input vectors (equation 6.12), it suffices to show that

$$R(\mathbf{R}) = \text{rs}(\mathbf{P}). \tag{6.16}$$

To begin, we also need to define the *nullspace* of a matrix.

Definition 6.5 *The* nullspace $N(\mathbf{A})$ *of a matrix* $\mathbf{A} \in \mathbb{R}^{i \times j}$ *is the set of vectors* $\mathbf{x}$ *defined as follows.*

$$N(\mathbf{A}) = \{\mathbf{x} \mid \mathbf{x} \in \mathbb{R}^j \text{ and } \mathbf{A}\mathbf{x} = \mathbf{0}\}. \tag{6.17}$$

Thus, if we consider $\mathbf{A}$ as a mapping, $N(\mathbf{A})$ is the set of all vectors that map to $\mathbf{0}$. It is a well-known result (see [117]) that if $\text{rank}(\mathbf{A}) = r$ then $N(\mathbf{A})$ is a subspace of $\mathbb{R}^j$ and $\dim(N(\mathbf{A})) = j - r$. Also, for any matrix $\mathbf{A}$,

$$R(\mathbf{A})^{\perp} = N(\mathbf{A}^T) \tag{6.18}$$

where $R(\mathbf{A})^{\perp}$ denotes the *orthogonal complement* of $R(\mathbf{A})$. As usual, the orthogonal complement $S^{\perp}$ of some subspace S of $\mathbb{R}^i$ is defined as,

$$S^{\perp} = \{\mathbf{y} \in \mathbb{R}^i \mid \mathbf{y}^T\mathbf{x} = 0 \text{ for all } \mathbf{x} \in S\}. \tag{6.19}$$

Lemma 6.6 *Let* $\mathbf{A} \in \mathbb{R}^{i \times j}$ *be any matrix. Then* $N(\mathbf{A}) = N(\mathbf{A}^T\mathbf{A})$.

Proof Clearly if $\mathbf{Ax} = \mathbf{0}$ for some vector $\mathbf{x}$ then $\mathbf{A}^T\mathbf{Ax} = \mathbf{0}$, and so $\mathbf{x} \in N(\mathbf{A}) \Longrightarrow \mathbf{x} \in N(\mathbf{A}^T\mathbf{A})$. Conversely (see Strang [143]), if $\mathbf{A}^T\mathbf{Ax} = \mathbf{0}$ for some vector $\mathbf{x}$ then,

$$\mathbf{x}^T\mathbf{A}^T\mathbf{Ax} = (\mathbf{Ax})^T\mathbf{Ax} = \|\mathbf{Ax}\|^2 = 0 \tag{6.20}$$

and hence $\mathbf{Ax} = \mathbf{0}$; this means that $\mathbf{x} \in N(\mathbf{A}^T\mathbf{A}) \Longrightarrow \mathbf{x} \in N(\mathbf{A})$ and the proof is complete. □

A simple corollary of this result is that $\text{rank}(\mathbf{A}) = \text{rank}(\mathbf{A}^T\mathbf{A})$; in the current scenario we therefore have $\text{rank}(\mathbf{P}) = \text{rank}(\mathbf{P}^T\mathbf{P}) = \text{rank}(\mathbf{R})$. Now, it is easily verified (see [117]) that $\text{rs}(\mathbf{P}^T\mathbf{P}) \subseteq \text{rs}(\mathbf{P})$, and consequently as $\text{rank}(\mathbf{P}^T\mathbf{P}) = \text{rank}(\mathbf{P})$ we must have $\text{rs}(\mathbf{R}) = \text{rs}(\mathbf{P})$. To see that this is true note that because $\text{rank}(\mathbf{P}) = \text{rank}(\mathbf{P}^T\mathbf{P})$ we have $\dim(\text{rs}(\mathbf{P})) = \dim(\text{rs}(\mathbf{P}^T\mathbf{P})) = r$. There therefore exists a linearly independent set $\{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_r\}$ of r vectors in $\text{rs}(\mathbf{P}^T\mathbf{P})$ which span $\text{rs}(\mathbf{P}^T\mathbf{P})$. However, because $\text{rs}(\mathbf{P}^T\mathbf{P}) \subseteq \text{rs}(\mathbf{P})$ these vectors also form a basis for $\text{rs}(\mathbf{P})$ (see [117]) and hence $\text{rs}(\mathbf{P}^T\mathbf{P}) = \text{rs}(\mathbf{P})$. Finally, as $\mathbf{R}$ is symmetric we have,

$$\begin{aligned} R(\mathbf{R}) &= \text{cs}(\mathbf{R}) \\ &= \text{rs}(\mathbf{R}) \\ &= \text{rs}(\mathbf{P}) \end{aligned} \tag{6.21}$$

which is precisely the result we set out to prove.

We now return to the normal equations. When $\mathbf{R}$ is rank deficient, and as we have seen, $\mathbf{p} \in R(\mathbf{R})$ such that the normal equations have an infinite number of solutions, the normal equations have a solution space defined as the set of all vectors of the form,

$$\mathbf{w}_{\text{opt}} = \mathbf{w}_{\text{sol}} + \sum_{i=1}^{(m+1)-r} \theta_i \mathbf{n}_i \tag{6.22}$$

where $\mathbf{w}_{\text{sol}}$ is any single solution of $\mathbf{Rw} = \mathbf{p}$, the θ_i are reals and the set of vectors $\{\mathbf{n}_1, \mathbf{n}_2, \ldots, \mathbf{n}_{(m+1)-r}\}$ is a basis of $N(\mathbf{R})$.

We now prove in full a further important result, which is again often hinted at, but not fully verified, in the literature.

Lemma 6.7 *Let* $\mathbf{Rw} = \mathbf{p}$ *be a set of normal equations where* $\mathbf{R}$ *is rank deficient and hence there are an infinite number of solutions. Then all solutions provide the same error* $\xi(\mathbf{w})$.

Proof Let,

$$\begin{aligned} \mathbf{w}_1 &= \mathbf{w}_{\text{sol}} + \sum_{i=1}^{(m+1)-r} \theta_i^{(1)} \mathbf{n}_i \\ \mathbf{w}_2 &= \mathbf{w}_{\text{sol}} + \sum_{i=1}^{(m+1)-r} \theta_i^{(2)} \mathbf{n}_i \end{aligned} \tag{6.23}$$

be any two solutions to the normal equations. Also, let the corresponding errors be $\xi_1 = \xi(\mathbf{w}_1)$ and $\xi_2 = \xi(\mathbf{w}_2)$. Then,

$$\begin{aligned} \xi_1 &= \sum_{i=1}^{k} [o_i - \mathbf{w}_1^T \tilde{\mathbf{x}}_i]^2 \\ \xi_2 &= \sum_{i=1}^{k} [o_i - \mathbf{w}_2^T \tilde{\mathbf{x}}_i]^2. \end{aligned} \tag{6.24}$$

The important terms in equation 6.24 are those of the form $\mathbf{w}_j^T\tilde{\mathbf{x}}_i$ and so we consider them separately. We have,

$$\begin{aligned}\mathbf{w}_j^T\tilde{\mathbf{x}}_i &= \left[\mathbf{w}_{\text{sol}} + \sum_{l=1}^{(m+1)-r} \theta_l^{(j)}\mathbf{n}_l\right]^T \tilde{\mathbf{x}}_i \\ &= \mathbf{w}_{\text{sol}}^T\tilde{\mathbf{x}}_i + \sum_{l=1}^{(m+1)-r} \theta_l^{(j)}\mathbf{n}_l^T\tilde{\mathbf{x}}_i \qquad (6.25)\end{aligned}$$

where $\mathbf{n}_l \in N(\mathbf{R})$ for $l = 1, 2, \ldots, (m+1) - r$. Now, from the results obtained above we have $N(\mathbf{R}) = R(\mathbf{R})^\perp = \text{rs}(\mathbf{P})^\perp$ and consequently we have $\mathbf{n}_l^T\tilde{\mathbf{x}}_i = 0$ for $l = 1, 2, \ldots, (m+1) - r$ and $i = 1, 2, \ldots, k$. Each term in the summation in equation 6.25 is therefore zero and we have,

$$\xi_1 = \xi_2 = \sum_{i=1}^{k}[o_i - \mathbf{w}_{\text{sol}}^T\tilde{\mathbf{x}}_i]^2. \qquad (6.26)$$

□

The Minimum Norm Solution

In the solution space defined by equation 6.22 there is a unique vector which has the minimum Euclidean norm.

Definition 6.8 *The minimum norm weight vector* $\mathbf{w}_{\text{min}}$ *is the weight vector the Euclidean norm* $\|\mathbf{w}_{\text{min}}\|$ *of which satisfies,*

$$\|\mathbf{w}_{\text{min}}\| = \min\{\|\mathbf{w}\| \mid \mathbf{Rw} = \mathbf{p}\}. \qquad (6.27)$$

In order to calculate $\mathbf{w}_{\text{min}}$ we use the *Moore-Penrose pseudoinverse* $\mathbf{R}^+$ of $\mathbf{R}$; the vector $\mathbf{w}_{\text{min}}$ is,

$$\mathbf{w}_{\text{min}} = \mathbf{R}^+\mathbf{p}. \qquad (6.28)$$

Henceforth, we will refer to $\mathbf{A}^+$ for some matrix $\mathbf{A}$ simply as the pseudoinverse of $\mathbf{A}$. In order to define the pseudoinverse of a matrix we now introduce the *singular value decomposition (SVD)*. The pseudoinverse can be defined without referring to the SVD, however we will need the latter in order to construct a self-structuring training algorithm below and so we now introduce the following result. For a complete treatment of the SVD see [142], or Klema and Laub [144].

Theorem 6.9 (Singular value decomposition) *Consider a matrix* $\mathbf{A} \in \mathbb{R}^{i\times j}$ *where* $\text{rank}(\mathbf{A}) = r$. *There exist orthogonal*[2] *matrices* $\mathbf{U} \in \mathbb{R}^{i\times i}$ *and* $\mathbf{V} \in \mathbb{R}^{j\times j}$ *such that,*

$$\mathbf{A} = \mathbf{U\Sigma V}^T. \qquad (6.29)$$

In equation 6.29,

$$\mathbf{\Sigma} = \begin{bmatrix} \mathbf{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \qquad (6.30)$$

where $\mathbf{S} = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r)$ *and where* $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$.

[2]In the sense that $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ and $\mathbf{V}^T\mathbf{V} = \mathbf{I}$.

The values $\sigma_1, \sigma_2, \ldots, \sigma_r$ along with $\sigma_{r+1} = 0, \sigma_{r+2} = 0, \ldots, \sigma_j = 0$ are the *singular values*[3] of $\mathbf{A}$. They are equal to the positive square roots of the eigenvalues of $\mathbf{A}^T\mathbf{A}$. The columns of $\mathbf{U}$ are the *left singular vectors*, and the columns of $\mathbf{V}$ the *right singular vectors* of $\mathbf{A}$. The matrices $\mathbf{U}$ and $\mathbf{V}$ are useful in characterizing the nullspace and range of a matrix. Let $\mathbf{U} = [\ \mathbf{u}_1 \ \ \mathbf{u}_2 \ \ \cdots \ \ \mathbf{u}_i\ ]$ and let $\mathbf{V} = [\ \mathbf{v}_1 \ \ \mathbf{v}_2 \ \ \cdots \ \ \mathbf{v}_j\ ]$. The columns of $\mathbf{U}$ with same-numbered singular values that are non-zero span the range of $\mathbf{A}$ and the columns of $\mathbf{V}$ with same-numbered singular values that are zero span the nullspace of $\mathbf{A}$. Also, note that as in the present case $\mathbf{R}$ is symmetric, we have $\mathbf{R}^T = \mathbf{R}$ and hence,

$$\mathbf{R}^T = (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^T = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T = \mathbf{R}. \tag{6.31}$$

Consequently we have $\mathbf{U} = \mathbf{V}$ and $\mathbf{U}$ alone provides a basis both for $R(\mathbf{R})$ and for $N(\mathbf{R})$. The SVD can be calculated in practice in a highly stable manner, and it is the only entirely reliable tool for dealing with rank deficient problems of this type in practice. It is for these reasons that we introduce it at an early stage, with a view to using it as the basis for a self-structuring training algorithm below.

Definition 6.10 (Pseudoinverse) *Let the matrix $\mathbf{A} \in \mathbb{R}^{i \times j}$ have rank r and singular value decomposition $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ where $\mathbf{\Sigma}$ is as defined in theorem 6.9. The pseudoinverse $\mathbf{A}^+$ of $\mathbf{A}$ is,*

$$\mathbf{A}^+ = \mathbf{V} \begin{bmatrix} \mathbf{S}' & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \mathbf{U}^T \tag{6.32}$$

where the matrix $\mathbf{S}'$ is defined as,

$$\mathbf{S}' = \operatorname{diag}\left(\frac{1}{\sigma_1}, \frac{1}{\sigma_2}, \ldots, \frac{1}{\sigma_r}\right). \tag{6.33}$$

We can now use the SVD to obtain an expression which describes the solution space and which can be calculated in practice. The solution space is the set of all vectors of the form,

$$\mathbf{w}_{\text{opt}} = \mathbf{R}^+\mathbf{p} + \sum_{i=r+1}^{m+1} \theta_i \mathbf{v}_i. \tag{6.34}$$

In equation 6.34 we have used $\mathbf{w}_{\text{min}}$ in place of $\mathbf{w}_{\text{sol}}$ as the latter can be any solution, and we have used the matrix $\mathbf{V}$ to provide a basis of $N(\mathbf{R})$. Again, the θ_i are real-valued; different values for the θ_i provide different solutions $\mathbf{w}_{\text{opt}}$.

The Use of the Minimum Norm Solution in Self-Structuring

As we mentioned briefly in the introduction to this chapter, perhaps the most commonly encountered technique for performing self-structuring involves beginning with a large network and attempting to reduce its size by considering the set of all weight vectors which provide an acceptable error on the training examples and penalizing weight vectors therein having relatively large Euclidean norms. Typically this is done by attempting to minimize an error $\xi'(\mathbf{w})$ of a form such as,

$$\xi'(\mathbf{w}) = \xi(\mathbf{w}) + \lambda \|\mathbf{w}\|^2, \tag{6.35}$$

[3]In numerical work using the SVD of a matrix $\mathbf{A}$ it is often necessary to decide whether a singular value is 'small enough' to be regarded as zero. A numerically stable way in which to make this decision is to regard a singular value as non-zero if it is greater than $(s \times \sigma_1 \times \text{fp})$, where s is the largest dimension of $\mathbf{A}$ and fp is the *floating point relative accuracy* [145]. This is the technique used in the experimental work presented later in this chapter.

where $\xi(\mathbf{w})$ is as defined above and $\lambda > 0$ is a real-valued term which allows us to set the importance of norm reduction relative to the minimization of $\xi(\mathbf{w})$. This approach to self-structuring is usually referred to in the literature as the *weight decay* approach.

The weight decay technique has been applied to various different types of network, and incorporates a basic assumption that the resulting vector is likely to have a larger number of zero elements than one obtained by simply attempting to minimize $\xi(\mathbf{w})$. In the context of the development presented above, note that, when we consider general Φ-nets, if we have an infinite number of solutions described by equation 6.34 then the solution that minimizes $\xi'(\mathbf{w})$ is $\mathbf{w}_{\text{min}}$, because as we have seen, all solutions correspond to the same error $\xi(\mathbf{w})$, and $\mathbf{w}_{\text{min}}$ is the unique minimizer of the term $\lambda\|\mathbf{w}\|^2$ when $\lambda > 0$. We now demonstrate that for general Φ-nets $\mathbf{w}_{\text{min}}$ is not necessarily the weight vector in the solution space having the smallest possible number of non-zero weights, and hence does not necessarily provide the best reduced size network $\mathcal{N}'$. We do this using a simple example.

Consider the following simple problem. We wish to train a $(2,2)$ polynomial discriminant function network using the following training examples.

$$\begin{array}{ll} \mathbf{x}_1^T = [\ 0 \quad 1\ ] & o_1 = 1 \\ \mathbf{x}_2^T = [\ 1 \quad 0\ ] & o_2 = -1 \\ \mathbf{x}_3^T = [\ 1 \quad 1.5\ ] & o_3 = 1 \\ \mathbf{x}_4^T = [\ 0.5 \quad 0.25\ ] & o_4 = -1. \end{array} \tag{6.36}$$

The matrix $\mathbf{R}$ in this case has size 6-by-6 and rank 4. The set of normal equations in this case has an infinite number of solutions, and the minimum norm solution is,

$$\mathbf{w}_{\text{min}}^T = [\ -0.8703 \quad -0.7076 \quad 0.5870 \quad 0.5779 \quad -1.1786 \quad 1.2833\ ]. \tag{6.37}$$

The minimum norm solution clearly has no zero elements, or in fact any elements which could reasonably be assumed to be close enough to zero that the corresponding basis functions can be removed without causing a significant decrease in the performance of the network. However, we also have the following three alternative solutions, each of which has two zero elements,

$$\begin{array}{rcl} \mathbf{w}_{\text{opt}}^T & = & [\ 0 \quad 0 \quad -3.1667 \quad -1.0000 \quad -1.7500 \quad 4.1667\ ] \\ \mathbf{w}_{\text{opt}}^T & = & [\ -0.8667 \quad -0.1333 \quad 0 \quad 0 \quad -1.4667 \quad 1.8667\ ] \\ \mathbf{w}_{\text{opt}}^T & = & [\ -0.3333 \quad -3.3333 \quad 1.3333 \quad 2.6667 \quad 0 \quad 0\ ]. \end{array} \tag{6.38}$$

These solutions were calculated using the self-structuring training algorithm which we develop below.

6.5.4 Why Isn't Minimizing the Norm a Good Approach?

Self-structuring techniques based on the minimization of an error of a form such as $\xi'(\mathbf{w})$ as defined in equation 6.35 were originally introduced on the basis of an intuition that at first glance appears quite sound: that the minimum norm solution is likely to be a good one. Having established that this is not necessarily the case it is interesting to ask why.

The explanation is quite simple. The term $\lambda\|\mathbf{w}\|^2$ places a relatively high penalty on a small increase in magnitude of an already large weight, and gives only a relatively small reward if a small weight is set to zero. Altering the term to $\lambda\|\mathbf{w}\|$ improves the situation slightly but does not solve the problem, because clearly in order to perform self-structuring we should use a term which is relatively insensitive to fluctuations in the magnitude of large weights but rewards

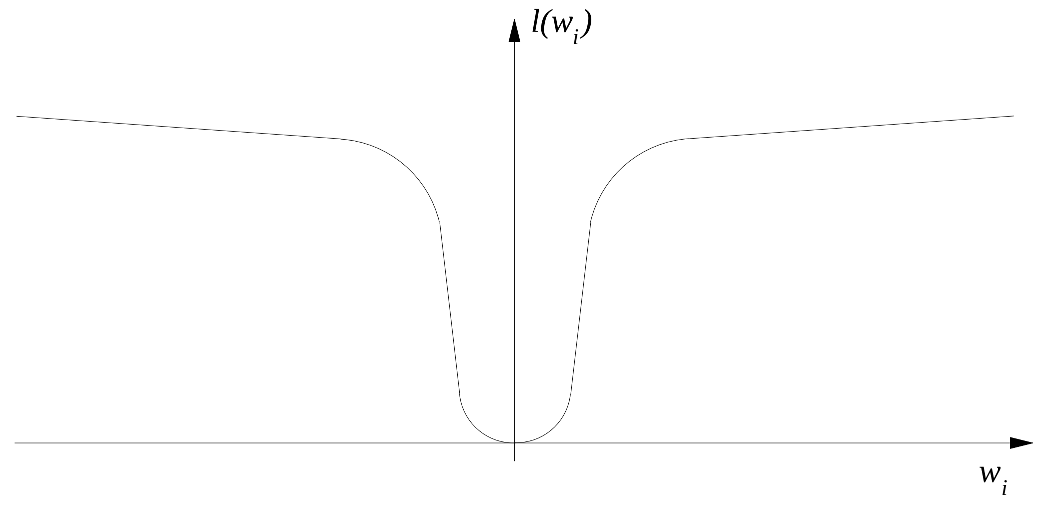


Figure 6.3: The general form of the function $l(w_i)$ used in our modification of the weight decay technique.

changes which set weights to zero. We therefore suggest an approach[4] in which we attempt to minimize an error $\xi''(\mathbf{w})$ of a form such as,

$$\xi''(\mathbf{w}) = \xi(\mathbf{w}) + \lambda R(\mathbf{w}), \tag{6.39}$$

where $\lambda > 0$ is again a real-valued term and the function $R(\mathbf{w})$ is of the form,

$$R(\mathbf{w}) = \sum_{i=0}^{m} l(w_i), \tag{6.40}$$

where w_i are as usual the weights in $\mathbf{w}$ and l is any function of the general form shown in figure 6.3. The optimum form for the function l would be,

$$l(w_i) = \begin{cases} c & \text{if } w_i \neq 0 \\ 0 & \text{if } w_i = 0 \end{cases}, \tag{6.41}$$

where c is a constant. However, in practice it may be preferable to use a function such as,

$$l(w_i) = \tanh(w_i^2), \tag{6.42}$$

especially if any form of gradient descent is to be used in order to minimize $\xi''(\mathbf{w})$; we pursue this approach in section 6.8.

Intuitively, we expect that by using an error of a form such as $\xi''(\mathbf{w})$ a training algorithm will be able to find better solutions than $\mathbf{w}_{\text{min}}$ because it will now be able to set weights to zero if this dictates that other weights take on values with relatively large magnitudes. Evidence that this ability is necessary can clearly be seen if we compare the minimum norm solution given in equation 6.37 for the simple example in the last subsection with the three alternative solutions given in equation 6.38, and also if we examine the experimental results provided in section 6.6.

Our criticism of the use of the term $\lambda\|\mathbf{w}\|^2$ extends readily to the case of networks other than Φ-nets, and for this reason we suggest that the use of this term will not in general provide a

[4]This modification to the weight decay technique has been suggested independently by Dr. M. R. Lynch.

good self-structuring technique. Similar criticisms of the simple weight decay approach to those advanced in this subsection have been made independently by Nowlan and Hinton [146]. To our knowledge, the simple modification that we have suggested in equations 6.39 and 6.40 has not previously been applied to the training of general Φ-nets.

6.5.5 A Better Solution

Let us briefly summarize the results of the analysis presented above. When training a Φ-net by selecting $\mathbf{w}$ in order to minimize the error $\xi(\mathbf{w})$ for the available training sequence T_k we need to solve a set of normal equations. If the relevant matrix $\mathbf{R}$ is non-singular then there is a single, unique solution and we cannot perform self-structuring without increasing $\xi(\mathbf{w})$. However, in certain realistic circumstances $\mathbf{R}$ is singular and in this case we have an infinite number of possible solutions each providing the same error; in this case the space of all solutions consists of the vectors of the form,

$$\mathbf{w}_{\text{opt}} = \mathbf{w}_{\text{min}} + \sum_{i=r+1}^{m+1} \theta_i \mathbf{v}_i. \tag{6.43}$$

We therefore appear to have some freedom to perform self-structuring because we can attempt to search for a $\mathbf{w}_{\text{opt}}$ containing many zero elements.

We saw earlier for a simple specific example that, contrary to the common assumption, $\mathbf{w}_{\text{min}}$ is not necessarily the weight vector having the largest possible number of zero elements, and that in fact it may have no zero elements at all. When addressing a given problem we must therefore ask the following question if we wish to perform self-structuring: is it possible to choose values for the θ_i in equation 6.43 such that some of the elements of the resulting $\mathbf{w}_{\text{opt}}$ (ideally as many as possible) are set to zero? We now show that it is always possible to obtain a vector $\mathbf{w}_{\text{opt}}$ which has at least $(W - r)$ zero elements where, as usual, W is the number of variable weights in $\mathbf{w}$ and $r = \text{rank}(\mathbf{R})$. This in fact follows directly from a standard result in linear algebra [117].

Lemma 6.11 *Let $\mathbf{Ab} = \mathbf{c}$ be a set of equations where $\mathbf{A} \in \mathbb{R}^{i \times i}$, $\mathbf{b} \in \mathbb{R}^i$, $\mathbf{c} \in \mathbb{R}^i$ and* $\text{rank}(\mathbf{A}) = j < i$. *Assume that the set of equations has an infinite number of solutions. Then there exists a set of $(i - j)$ elements of* $\mathbf{b}$ *which can be given* arbitrary *values, such that the remaining elements of* $\mathbf{b}$ *can then be assigned values which make* $\mathbf{b}$ *a solution of the set of equations.*

Our result follows directly from this lemma when we use zeros for the arbitrary values. Note that when we calculate the remaining weights, having set a subset of them to zero, we may obtain further zero weights as well. The important question is now that of how to find the elements that can be set arbitrarily in the manner stated. Although we can in principle find a suitable set of elements using a method involving the transformation of $\mathbf{R}$ into an *echelon matrix* [117] we will introduce in the next section an alternative algorithm which has two major advantages over this approach: firstly, it is based on the use of the singular value decomposition which, as we have already noted, is the only entirely reliable numerical tool for dealing with rank-deficient problems in practice and, secondly, it can often find more than one suitable set of elements, and it can also in general find one or more alternative, suboptimal sets of less than $(W - r)$ elements that can be set to zero.

6.6 A Self-Structuring Training Algorithm

In this section we use the theory developed above to construct a self-structuring training algorithm which we then test experimentally. We begin by developing the algorithm, which is based on lemma 6.11. We then present and discuss some experimental results.

6.6.1 The Self-Structuring Training Algorithm

Recall that, because $\mathbf{R}$ is symmetric, we can write its SVD as,

$$\mathbf{R} = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^T \tag{6.44}$$

where $\mathbf{\Sigma}$ and $\mathbf{U}$ are as defined above and $\mathbf{U}$ can be written using its columns as,

$$\mathbf{U} = [\ \mathbf{u}_1 \quad \mathbf{u}_2 \quad \cdots \quad \mathbf{u}_W\ ]. \tag{6.45}$$

We also know that $\text{span}\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_r\} = R(\mathbf{R})$ and $\text{span}\{\mathbf{u}_{r+1}, \mathbf{u}_{r+2}, \ldots, \mathbf{u}_W\} = N(\mathbf{R})$, and we can therefore express the solution space S as the set of all vectors,

$$\begin{aligned}\mathbf{w}_{\text{opt}} &= \mathbf{w}_{\text{min}} + \sum_{i=r+1}^{W} \theta_i \mathbf{u}_i \\ &= \mathbf{w}_{\text{min}} + \mathbf{N}\boldsymbol{\theta}\end{aligned} \tag{6.46}$$

where $\mathbf{N}$ is the matrix defined in terms of its columns and rows as,

$$\mathbf{N} = \left[\ \mathbf{u}_{r+1} \quad \mathbf{u}_{r+2} \quad \cdots \quad \mathbf{u}_W\ \right] = \begin{bmatrix} \mathbf{n}_1^T \\ \mathbf{n}_2^T \\ \vdots \\ \mathbf{n}_W^T \end{bmatrix} \tag{6.47}$$

and $\boldsymbol{\theta}^T = [\ \theta_{r+1} \quad \theta_{r+2} \quad \cdots \quad \theta_W\ ]$. Let $I = \{i_1, i_2, \ldots, i_{(W-r)}\} \subseteq \{1, 2, \ldots, W\}$ be a set containing the indices of the elements of $\mathbf{w}_{\text{opt}}$ that can be set arbitrarily, and which we will set to zero, and let $\mathbf{w}_{\text{min}}$ be written as,

$$\mathbf{w}_{\text{min}}^T = [\ w_{\text{min}}^{(1)} \quad w_{\text{min}}^{(2)} \quad \cdots \quad w_{\text{min}}^{(W)}\ ]. \tag{6.48}$$

We can think of the task of setting the arbitrary elements of $\mathbf{w}_{\text{opt}}$ to zero as one of choosing $\boldsymbol{\theta}$ such that the vector $\mathbf{N}\boldsymbol{\theta}$ 'cancels' the corresponding elements of $\mathbf{w}_{\text{min}}$; in other words, lemma 6.11 tells us that the set of equations,

$$\mathbf{N}'\boldsymbol{\theta} = \begin{bmatrix} \mathbf{n}_{i_1}^T \\ \mathbf{n}_{i_2}^T \\ \vdots \\ \mathbf{n}_{i_{(W-r)}}^T \end{bmatrix} \boldsymbol{\theta} = - \begin{bmatrix} w_{\text{min}}^{(i_1)} \\ w_{\text{min}}^{(i_2)} \\ \vdots \\ w_{\text{min}}^{(i_{(W-r)})} \end{bmatrix} = -\mathbf{w}'_{\text{min}} \tag{6.49}$$

is consistent. Our task is to find a set I which leads to a consistent set of equations of this type.

The obvious way in which to search for a suitable set I is to search over all $(W-r)$-element subsets of the set $\{1, 2, \ldots, W\}$, constructing the set of equations described by equation 6.49 and testing for consistency until we find a suitable subset. However, as there are $C_{(W-r)}^{W}$ such subsets this is not in general a computationally feasible approach. We now suggest an alternative algorithm which is computationally feasible; this algorithm unfortunately does not guarantee to find a set of $(W-r)$ zero weights. However, in practice it is very successful in doing so, and furthermore, if it is unable to find such a set it attempts to return a smaller set of weights that can be set to zero.

We first describe the algorithm informally. Beginning with a set $I' = \{1, 2, \ldots, W\}$ of available indices we attempt to construct a set of consistent equations as in equation 6.49 by beginning with a single equation,

$$\mathbf{n}_{i_1}^T \boldsymbol{\theta} = -w_{\text{min}}^{(i_1)} \tag{6.50}$$

for an arbitrary index $i_1 \in I'$ (this equation clearly always has a solution) and adding further equations one at a time such that the overall set of equations is always consistent. If at any time we obtain a consistent set of $(W - r)$ equations then we have a valid set I. Alternatively, if we find that a set of less than $(W - r)$ equations cannot be added to while still being consistent we put aside the set of indices used so far, as we can use them later if necessary as a suboptimal solution (we also store such a suboptimal solution if we exhaust the supply of available indices). After finding a valid set I or a suboptimal solution we delete the indices used from the set I' and begin the process again using the reduced set I' of available indices. This process continues until I' is empty. We can think of this process as one of forming a partition of the set I'; each equivalence class in the partition contains a set of indices which either provides a valid set I or a suboptimal solution.

In order to make the operation of this algorithm clearer, we now provide a pseudo-code version. In the following, '`FINISHED`' is a Boolean variable which can take the value '`TRUE`' or '`FALSE`', T is a set used for temporary storage of indices and I' is a set containing available indices. The pseudo-code is as follows.

```
Begin
    Let $I' = \{1, 2, \ldots, W\}$ and let $T = \emptyset$.
    While $I' \neq \emptyset$
        Choose an arbitrary $i_1 \in I'$ and let $T = \{i_1\}$.
        Let FINISHED = FALSE.
        While ($I' \setminus T \neq \emptyset$) and ($| T | < W - r$) and (FINISHED = FALSE)
            Search through the indices $i \in I' \setminus T$. One of the following must occur:
                (a) We find an $i$ such that on adding the corresponding
                equation to the set of equations given by $T$ we obtain
                a new set which is consistent. In this case let $T = T \cup i$.

                (b) We find that no $i \in I' \setminus T$ lets us add a new equation
                such that the new set is consistent. In this case let
                FINISHED = TRUE.
        End
        Store $T$ and let $I' = I' \setminus T$.
    End
End
```

In the implementation of this algorithm used in the experiments described below, i_1 is always chosen as the smallest element of I', and the search through the indices in $I' \setminus T$ is by numerical order. Note that, if at some point during the construction of a set of equations it is found that some index cannot be included in the set T because the resulting set of equations is inconsistent, then it is not necessary to attempt to include this index again at a later stage in the construction of this particular set of equations, as clearly the attempt must be unsuccessful.

6.6.2 Experiments with the Self-Structuring Training Algorithm

We now experimentally test the training algorithm developed above using two problems: the well-known *parity problem* and the *two-circles problem*. We have chosen these problems because when they are addressed using appropriate polynomial discriminant function Φ-nets, that is, (n, d) discriminators which use all possible basis functions up to an appropriate degree d, there is in each case a known weight vector which solves the problem and which has only a small number (one or two in the problems of interest) of non-zero weights[5]. Consequently, whenever we have

[5]The use of these particular problems was suggested by Dr. P. Rayner.

Experiment number	n, d, W, r	Number of examples per class	Number of non-zero weights in the different solutions		
			Minimum norm	Semi-optimum	Non-optimum
1	2,6,28,4	2	6	2	4
2	2,8,45,4	2	10	2	8
3	4,4,70,16	8	1	1	1,$\star$
4	4,5,126,16	8	1	1	$\star$,1
5	4,4,70,12	6	12	1	10
6	4,5,126,12	6	26	3	23

Table 6.1: Results obtained for various instances of the parity problem using various different networks. 'Semi-optimum' solutions are those found by our algorithm which are known to have at least $(W - r)$ zero weights. 'Non-optimum' solutions are other solutions found by our algorithm. All weights having a magnitude of less than or equal to 10^{-11} were regarded as zero. See the main text for an explanation of the '$\star$' entries.

an infinite number of possible solutions $\mathbf{w}_{\text{opt}}$ we can test the ability of the algorithm to find a known optimum solution corresponding to a small Φ-net.

In the following we will refer to the weight vectors which are known to be the optimum solutions to the specific problems addressed as *true optimum solutions*. We will refer to weight vectors produced by our algorithm which are known to have at least $(W - r)$ zero weights as *semi-optimum solutions*, and to other weight vectors produced by our algorithm as *non-optimum solutions*. Note that 'optimum' in this case refers to the quality of a solution in terms of the size of the corresponding network; semi-optimum solutions and non-optimum solutions have the same error $\xi(\mathbf{w})$. Note also that this nomenclature is intended only as a means for identifying different types of solution; in some cases, as we shall see, non-optimum solutions can be better, in terms of network size, than semi-optimum solutions, and either can be as good as a true optimum solution.

The Parity Problem

We first test our algorithm using an (n, d) discriminator applied to the standard parity problem. In this problem, inputs to the network are drawn from the set $\{-1, +1\}^n$, and an input should be assigned to class one if it contains an even number of +1s and to class two otherwise. If, as usual, the index for class one is +1 and that for class two is −1, and if n is even, then we can solve the problem using a (n, d) discriminator having only a single non-zero weight provided $d \geq n$ because clearly the function,

$$f(\mathbf{x}) = \prod_{i=1}^{n} x_i \tag{6.51}$$

solves the problem perfectly. Also, as there are 2^n possible input vectors and C_n^{n+d} weights in the network we will in general have more weights than possible training examples and consequently there will be an infinite number of possible solutions.

Table 6.1 summarizes the results obtained when using our self-structuring training algorithm to train various different polynomial discriminant function Φ-nets for various instances of the parity problem. Results obtained by calculating the minimum norm weight vector in each case are also included. Weights were regarded as non-zero if they had magnitude greater than 10^{-11}; it was straightforward in the cases examined to decide a good point at which to discriminate between zero and non-zero weights, as the smallest weight magnitude of a non-zero weight under this criterion was approximately 0.05. In all cases the semi-optimum solutions found by our algorithm have at least $(W - r)$ zero weights as expected. In all cases our algorithm produces a network which is either as small as, or smaller than that obtained by calculating the minimum norm solution, and in experiments 3, 4 and 5 it finds the known true optimum solution. The most interesting result is obtained in experiment 5, in which our algorithm finds a solution which corresponds to the known true optimum network, whereas the minimum norm solution corresponds to a larger network.

In each of experiments 3 and 4 our algorithm obtains an important non-optimum result; these results are marked in table 6.1 using '$\star$'. Recall that this particular problem can be solved in all the cases examined using only a single weight which has value 1 (equation 6.51). In experiment 3 our algorithm tries while searching for a semi-optimum solution to set the weight corresponding to the true optimum solution to zero along with others which have already been found during its search, but finds that it cannot; eventually it constructs a non-optimum solution which guarantees only to set this single weight to zero. In experiment 4 our algorithm at one point attempts to begin searching for a new semi-optimum solution by setting the weight corresponding to the known true optimum solution to zero as its starting point, but then finds that none of the remaining available weights can be set to zero concurrently and thus again obtains a non-optimum solution which only guarantees to set this single weight to zero. We know from equation 6.50 that it is always possible to set any single weight to zero, however in the two special solutions described we are setting the weight to zero which would equal 1 in the true optimum solution, and the consequence of this is that some of the remaining weights in the solution have very large magnitudes (approximately 2×10^{16} in experiment 3 and 9×10^{15} in experiment 4). In fact in these two cases the solution calculated suffers from numerical error to the extent that it is an inaccurate solution to the normal equations. We discuss further below the potential problem of requiring weights with large magnitudes.

The Two-Circles Problem

An instance of the two-circles problem is shown in figure 6.4. In this problem inputs to the network are in $\mathbb{R}^2$ and classes one and two correspond to two concentric circles, centred at the origin and having different radii. In the following, class one has radius $\frac{1}{2}$ and class two has radius 1, and training examples are chosen at random such that they are uniformly distributed around the circles. Clearly, provided the class indices are chosen correctly (in this case $o_i = 0.25$ and $o_i = 1$ for classes one and two respectively) this problem can be solved using a $(2, d)$ discriminator provided $d \geq 2$ because the function,

$$f(\mathbf{x}) = x_1^2 + x_2^2 \tag{6.52}$$

solves the problem perfectly.

Table 6.2 summarizes the results obtained when using our self-structuring training algorithm to train various different polynomial discriminant function Φ-nets for various different instances of the two-circles problem. Results obtained by calculating the minimum norm weight vector in each case are also included. In all cases $\mathbf{R}$ is rank deficient and consequently we always have an infinite number of solutions to the normal equations. Weights were regarded as non-zero if they had magnitude greater than 10^{-3}. In all cases the semi-optimum solutions found by our algorithm have at least $(W - r)$ zero weights as expected. The most important conclusion that can be drawn from table 6.2 is that our algorithm produces smaller networks than those obtained using the minimum norm solution in all of the 14 cases. Unfortunately however, in no case do we obtain the known, true optimum solution.

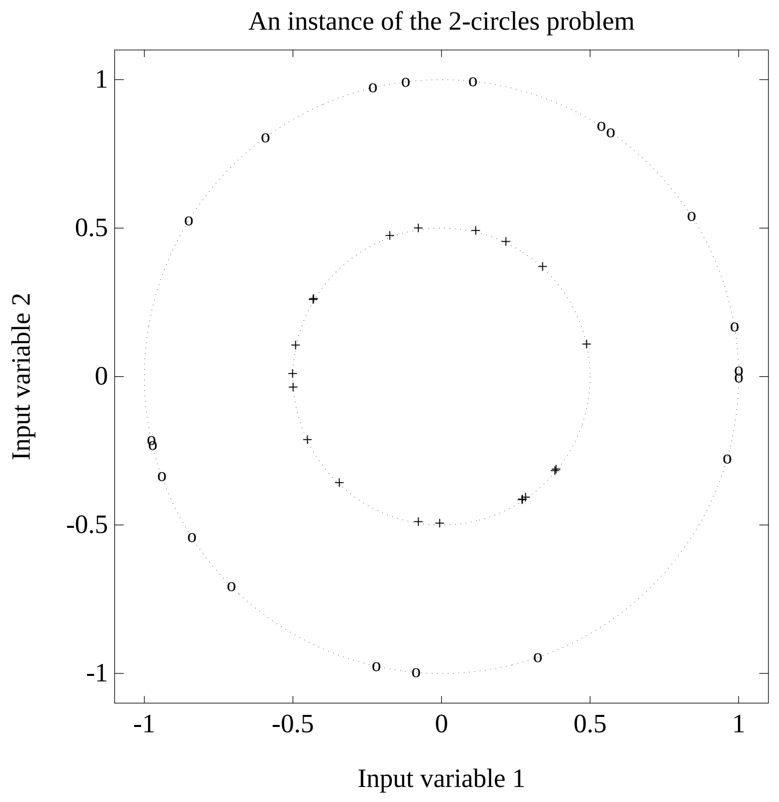


Figure 6.4: An instance of the two-circles problem. Inputs in class one lie on a circle centred at the origin and having radius $\frac{1}{2}$, and inputs in class two lie on a circle centred at the origin and having radius 1. Twenty examples for each class are shown.

Experiment number	d, W, r	Number of examples per class	Number of non-zero weights in the different solutions		
			Minimum norm	Semi-optimum	Non-optimum
1	6,28,20	10	26	7,7,6	24
2	7,36,20	10	35	20,6	32
3	8,45,20	10	45	20	25
4	9,55,20	10	55	20	34
5	10,66,20	10	66	20	46
6	8,45,30	20	15	9,18,28	–
7	9,55,34	20	15	9,28	42
8	10,66,37	20	61	11,36	15
9	11,78,37	20	58	37	39
10	12,91,39	20	59	34	52
11	12,91,46	30	27	20,45	24
12	12,91,46	40	27	13,45	25
13	12,91,46	50	26	13	45,24
14	12,91,46	100	26	13,45	26

Table 6.2: Results obtained for various instances of the two-circles problem using various different networks. 'Semi-optimum' solutions are those found by our algorithm which are known to have at least $(W - r)$ zero weights. 'Non-optimum' solutions are other solutions found by our algorithm. All weights having a magnitude of less than or equal to 10^{-3} were regarded as zero.

Unlike in the case of the parity problem, the point at which to discriminate between zero and non-zero weights was not obvious in most of the cases studied here. We also examined the results obtained when using values of 10^{-6}, 10^{-4} and 10^{-2} instead of 10^{-3}; although we do not at present have a systematic method for choosing these values, the values used are appropriate as none of the weights in any of the minimum norm solutions for the cases examined had magnitude greater than 0.6 and as the values used often provide values of precisely $(W - r)$ for the numbers of zero weights in the semi-optimum solutions provided by our algorithm. For a value of 10^{-4} we again found that in all of the fourteen experiments our algorithm produces a smaller network than we can obtain using the minimum norm solution, and for values of 10^{-6} and 10^{-2} our algorithm produces the smaller network in all but two of the experiments.

One possible drawback of our approach is that, as we might expect from the discussion in subsection 6.5.4, the cost of obtaining a smaller network can be that this network needs to use weights having relatively large magnitudes. An example is shown in figure 6.5 in which we show the minimum norm weight vector obtained in experiment 4 alongside the semi-optimum solution obtained by our algorithm in the same experiment. The exceptional results obtained when addressing the parity problem are extreme examples of this effect. We suggest that a simple way to prevent this problem from arising, in any case in which certain basis functions are known to be important and consequently should not be removed, is to restrict the search for sets of $(W - r)$ zero weights to sets which do not include weights corresponding to these important basis functions.

6.6.3 How Hard is it to Find a Set of $(W - r)$ Zero Weights?

On the basis of the theoretical development of the self-structuring training algorithm above, an obvious question to ask is whether all possible sets of $(W - r)$ weights can be set arbitrarily. Our

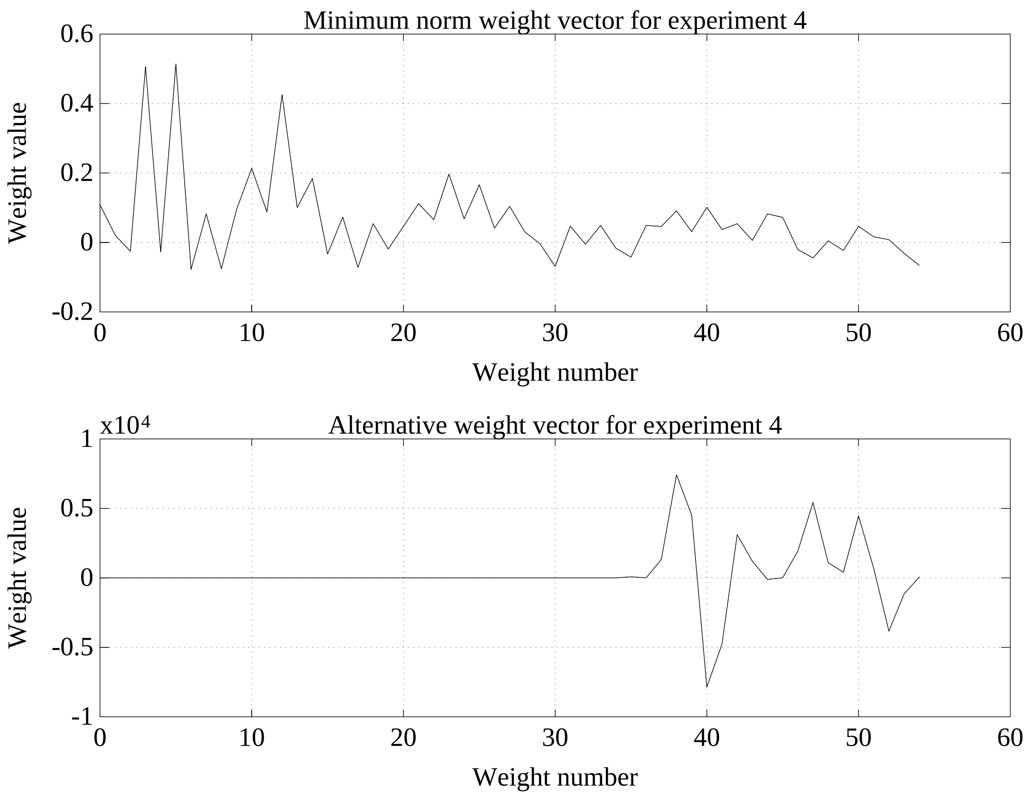


Figure 6.5: Use of a small network may be at the expense of the need for weights having relatively large magnitudes. The upper graph shows the minimum norm weight vector calculated in experiment 4, and the lower graph shows the weight vector for the semi-optimum solution calculated by our algorithm in the same experiment, which allows us to use a network having fewer weights. However, the magnitudes of some of these weights are clearly much larger.

experimental results for the parity problem show that this is not in fact the case. However, our experience in applying the algorithm suggests that sets of $(W-r)$ weights that cannot be set to zero are the exception, rather than the rule. This is a highly desirable property as it suggests that our algorithm will in general be successful in finding a suitable set of weights.

6.7 Removing Further Basis Functions

Having constructed and tested a self-structuring training algorithm, we now make a return to our theoretical analysis and address a further relevant question: by how much do we increase the error $\xi(\mathbf{w})$ if we attempt to force more than the maximum possible number of weights (as dictated by the solutions of the form of equation 6.43) to be zero?

When we calculate the SVD of $\mathbf{R}$ in practice it is often found that some singular values are close to, but not in fact equal to zero[6]. For example, if we take an instance of the two-circles problem having 20 examples in each class and construct $\mathbf{R}$ for an (n, d) polynomial discriminator with $d = 8$ then we find that $\mathbf{R}$ has size 45×45 and $\text{rank}(\mathbf{R}) = 30$. However, only 24 of the singular values of $\mathbf{R}$ are larger than 10^{-4}. If zero mean Gaussian noise with variance 10^{-3} is added to the original data then $\mathbf{R}$ has rank 40 but only 28 of its singular values are larger than 10^{-4}. As the solution space is constructed using columns of the matrix $\mathbf{V}$ which correspond to zero singular values it is natural to ask what happens if we augment the solution space using further columns of $\mathbf{V}$ which correspond to these small singular values. Intuitively, we would like if possible to obtain a further reduction in the size of our network, in addition to any obtained using the standard solution space, by searching in the new space having an increased 'size'. However, by using a solution $\mathbf{w}_{\text{aug}}$ from the new, augmented space we will increase the error $\xi(\mathbf{w}_{\text{aug}})$ of the resulting network, and we now address the question of by precisely how much the error will be increased. Although intuitively we would like to use additional columns of $\mathbf{V}$ corresponding to small singular values (and the following analysis indicates that this is a sound intuition) our work below applies regardless of which additional columns of $\mathbf{V}$ are used.

We can now define the *augmented solution space* S' as the set of all vectors of the form,

$$\mathbf{w}_{\text{aug}} = \mathbf{w}_{\text{min}} + \sum_{i=r+1}^{W} \theta_i \mathbf{u}_i + \sum_{i \in I} \lambda_i \mathbf{u}_i. \tag{6.53}$$

In equation 6.53, $I \subseteq \{1, 2, \ldots, r\}$ is a set containing the indices of the additional columns of $\mathbf{U}$ used and the λ_i are reals[7] for $i \in I$. Clearly, $S \subseteq S'$. We will now prove the following result which tells us about the error $\xi(\mathbf{w}_{\text{aug}})$ obtained if we use a weight vector $\mathbf{w}_{\text{aug}} \in S'$.

Theorem 6.12 *Let σ_i where $i = 1, 2, \ldots, W$ be the singular values of $\mathbf{R}$ where $\sigma_i > 0$ for $i = 1, 2, \ldots, r$ and $\sigma_i = 0$ for $i = r+1, r+2, \ldots, W$. Then the error $\xi(\mathbf{w}_{\text{aug}})$ obtained using a weight vector from the augmented solution space S' is,*

$$\xi(\mathbf{w}_{\text{aug}}) = \xi(\mathbf{w}_{\text{opt}}) + \sum_{i \in I} \lambda_i^2 \sigma_i, \tag{6.54}$$

where I is the set defined above.

Proof Beginning with any vector $\mathbf{w}_{\text{aug}} \in S'$ we have,

$$\xi(\mathbf{w}_{\text{aug}}) = \sum_{j=1}^{k} [o_j - \mathbf{w}_{\text{aug}}^T \tilde{\mathbf{x}}_j]^2$$

[6]This was originally pointed out to the author by Dr. P. Rayner.

[7]Note that the λ_i used here are not related to the parameter λ introduced in section 6.5.

$$
\begin{aligned}
&= \sum_{j=1}^{k}\left[o_j - \left(\mathbf{w}_{\text{min}} + \sum_{l=r+1}^{W}\theta_l\mathbf{u}_l + \sum_{i\in I}\lambda_i\mathbf{u}_i\right)^T\tilde{\mathbf{x}}_j\right]^2 \\
&= \sum_{j=1}^{k}\left[o_j - \left(\mathbf{w}_{\text{min}} + \sum_{l=r+1}^{W}\theta_l\mathbf{u}_l\right)^T\tilde{\mathbf{x}}_j - \left(\sum_{i\in I}\lambda_i\mathbf{u}_i^T\right)\tilde{\mathbf{x}}_j\right]^2 \\
&= \sum_{j=1}^{k}\left[(o_j - \mathbf{w}_{\text{opt}}^T\tilde{\mathbf{x}}_j) - \left(\sum_{i\in I}\lambda_i\mathbf{u}_i^T\right)\tilde{\mathbf{x}}_j\right]^2. \qquad (6.55)
\end{aligned}
$$

Applying lemma 6.7 we obtain,

$$
\xi(\mathbf{w}_{\text{aug}}) = \xi(\mathbf{w}_{\text{opt}}) - 2\sum_{j=1}^{k}(o_j - \mathbf{w}_{\text{opt}}^T\tilde{\mathbf{x}}_j)\left(\sum_{i\in I}\lambda_i\mathbf{u}_i^T\tilde{\mathbf{x}}_j\right) + \sum_{j=1}^{k}\left(\sum_{i\in I}\lambda_i\mathbf{u}_i^T\tilde{\mathbf{x}}_j\right)^2. \qquad (6.56)
$$

We now deal with equation 6.56 in two separate parts. Firstly, we have,

$$
\begin{aligned}
2\sum_{j=1}^{k}(o_j - \mathbf{w}_{\text{opt}}^T\tilde{\mathbf{x}}_j)\left(\sum_{i\in I}\lambda_i\mathbf{u}_i^T\tilde{\mathbf{x}}_j\right) &= 2\left(\sum_{i\in I}\lambda_i\mathbf{u}_i^T\right)\left(\sum_{j=1}^{k}o_j\tilde{\mathbf{x}}_j - \sum_{j=1}^{k}\tilde{\mathbf{x}}_j\tilde{\mathbf{x}}_j^T\mathbf{w}_{\text{opt}}\right) \\
&= 2\left(\sum_{i\in I}\lambda_i\mathbf{u}_i^T\right)\left(\mathbf{p} - \mathbf{R}\mathbf{w}_{\text{opt}}\right) \\
&= 0 \qquad (6.57)
\end{aligned}
$$

where the final step follows because $\mathbf{w}_{\text{opt}}$ is a solution to the normal equations. We must now consider the term,

$$
\sum_{j=1}^{k}\left(\sum_{i\in I}\lambda_i\mathbf{u}_i^T\tilde{\mathbf{x}}_j\right)^2 = \sum_{j=1}^{k}(\lambda_{i_1}\mathbf{u}_{i_1}^T\tilde{\mathbf{x}}_j + \lambda_{i_2}\mathbf{u}_{i_2}^T\tilde{\mathbf{x}}_j + \cdots + \lambda_{i_q}\mathbf{u}_{i_q}^T\tilde{\mathbf{x}}_j)^2 \qquad (6.58)
$$

where the indices $i_1, i_2, \ldots, i_q$ are the members of I and $q =\mid I \mid$. This can be written as a sum of terms of the form,

$$
\begin{aligned}
\sum_{j=1}^{k}(\lambda_a\mathbf{u}_a^T\tilde{\mathbf{x}}_j)(\lambda_b\mathbf{u}_b^T\tilde{\mathbf{x}}_j) &= \lambda_a\lambda_b\sum_{j=1}^{k}\mathbf{u}_a^T\tilde{\mathbf{x}}_j\tilde{\mathbf{x}}_j^T\mathbf{u}_b \\
&= \lambda_a\lambda_b\mathbf{u}_a^T\mathbf{R}\mathbf{u}_b \\
&= \lambda_a\lambda_b\mathbf{u}_a^T\mathbf{U}\boldsymbol{\Sigma}\mathbf{U}^T\mathbf{u}_b \qquad (6.59)
\end{aligned}
$$

where $a \in I$ and $b \in I$. Because $\mathbf{U}$ is an orthogonal matrix we have $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ and it is therefore easily shown that,

$$
\lambda_a\lambda_b\mathbf{u}_a^T\mathbf{U}\boldsymbol{\Sigma}\mathbf{U}^T\mathbf{u}_b = \begin{cases} 0 & \text{if } a \neq b \\ \lambda_a^2\sigma_a & \text{if } a = b \end{cases}. \qquad (6.60)
$$

We therefore have,

$$
\sum_{j=1}^{k}\left(\sum_{i\in I}\lambda_i\mathbf{u}_i^T\tilde{\mathbf{x}}_j\right)^2 = \sum_{i\in I}\lambda_i^2\sigma_i. \qquad (6.61)
$$

Combining equations 6.56, 6.57 and 6.61 we obtain the required result. □

Theorem 6.12 tells us that by augmenting the solution space in the manner suggested we may be able to obtain further reductions in the size of our network without causing a significant increase in the error obtained. Specifically, provided we form S' using extra columns of $\mathbf{U}$ corresponding to small singular values, and the magnitudes of the λ_i required are not too large, then we can cause only small increases of size $\sum_{i \in I} \lambda_i^2 \sigma_i$ in the error obtained. The alternative situation, in which we use columns of $\mathbf{U}$ corresponding to large singular values but insist that the magnitudes of the λ_i are small, is rather less relevant as in this case we can obviously only make small changes to an existing solution $\mathbf{w}_{\text{opt}}$.

This result generalizes easily to the case in which $\mathbf{R}$ is non-singular, but has some small singular values. For example, if we again construct $\mathbf{R}$ for a $(2,8)$ polynomial discriminator, using training examples for the two-spirals problem with 100 examples in each class and additive zero mean Gaussian noise with variance 0.001 (see figure 6.1), then $\mathbf{R}$ is non-singular ($\text{rank}(\mathbf{R}) = 45$). However, only 28 of its singular values are greater than 10^{-2} and only 36 are greater than 10^{-3}. If $\mathbf{R}$ is non-singular then there is a unique solution $\mathbf{w} = \mathbf{R}^{-1}\mathbf{p}$ to the normal equations, and in this case we can form an augmented solution space containing vectors of the form,

$$\mathbf{w}_{\text{aug}} = \mathbf{w} + \sum_{i \in I} \lambda_i \mathbf{u}_i \tag{6.62}$$

where $I \subseteq \{1, 2, \ldots, W\}$ is a set containing the indices of small singular values. Using a proof exactly analogous to that of theorem 6.12, it is easily verified that in this case we have,

$$\xi(\mathbf{w}_{\text{aug}}) = \xi(\mathbf{w}) + \sum_{i \in I} \lambda_i^2 \sigma_i. \tag{6.63}$$

6.8 Discussion

6.8.1 Selecting the Initial Set Φ

An important point that should be remembered when applying our approach to self-structuring is that the error $\xi(\mathbf{w})$ that will be obtained depends on the initial set Φ of basis functions used, from which we attempt to remove basis functions in order to obtain a smaller network. In general, this error will decrease as Φ is made larger, which suggests that it may be sensible to start with as large a set Φ as possible. Although this approach may be effective if the training examples are relatively noise free, it should be used with great care if they are noisy in order to avoid overfitting.

The technique introduced in section 6.7, which allows us to make small increases in $\xi(\mathbf{w})$ by removing certain basis functions provides a potential method for reducing overfitting in the latter case. An experimental evaluation of this technique provides a subject for future research.

6.8.2 Improvements to Our Approach

At present our self-structuring training algorithm needs to compute the singular value decomposition of the matrix $\mathbf{R}$. This step has two drawbacks: it can be time consuming when $\mathbf{R}$ is large and it requires that all the relevant training examples are available before training begins. It may be possible to improve matters here using SVD updating algorithms such as those of Bunch and Nielsen [147] or Moonen [148].

It may be possible to improve the numerical accuracy of our approach by calculating solutions on the basis of the set of equations $\mathbf{Pw} = \mathbf{o}$, rather than the normal equations [26]. It would also be useful to obtain systematic techniques for deciding how small a weight should be such that it can be regarded as zero, and for dealing with the potential problem of weights having large magnitudes which we mentioned above.

Finally, at present we can only guide our algorithm's search for sets of weights that can be set to zero if we know in advance that some basis functions are, or are not desirable. We also cannot in fact guarantee that the algorithm will always be successful in finding a set of $(W - r)$ such weights, although our experience suggests that it would be very unusual for the algorithm not to. We mentioned above that it is in principle possible to apply a technique which converts $\mathbf{R}$ into an echelon matrix in order to guarantee that a suitable set of $(W - r)$ weights is found, and we suggest that it may be possible to incorporate such a technique into our algorithm.

6.8.3 Classification Problems

In this chapter we have concentrated on networks having real-valued outputs, that is, for which $\mathcal{C} = \mathbb{R}$. Any two-class pattern classification problem having $\mathcal{C} = \mathbb{B}$ can clearly also be addressed using the methods developed in this chapter because we can simply train a network having real-valued outputs, using arbitrary positive real-valued class indices for patterns in class 1 and arbitrary negative real-valued class indices for patterns in class 2. There is clearly a large degree of freedom in the way in which we choose appropriate real-valued class indices, and the *Ho-Kashyap Procedure* [4] provides an automatic method for making this choice. A technique for the automatic choice of class indices introduced by Rayner and Lynch [36] may also be applicable here.

6.8.4 Other Approaches

Chen *et al.* [149, 150] have also introduced a method which can be used to determine a small architecture for certain types of Φ-net. Furthermore, it should be possible to perform self-structuring using *subset selection* techniques from the statistical literature, see for example [142] and Young [151], and it would be interesting to apply the Bayesian framework presented in [102, 103] to self-structuring for Φ-nets. The experimental comparison of alternative approaches such as these with the algorithm developed in this chapter forms an obvious, and important subject for future research.

6.8.5 The Modified Weight Decay Technique

In subsection 6.5.4 we suggested a method for modifying the error measure used in the simple weight decay technique. We have conducted some initial experiments on the basis of this idea by devising a modified form of the Least Mean Squares (LMS) algorithm [27].

The modified algorithm is derived as follows. We define the error $\epsilon_i(\mathbf{w})$ for the ith training example as,

$$\epsilon_i(\mathbf{w}) = [o_i - f_{\mathbf{w}}(\mathbf{x}_i)]^2 + \lambda R(\mathbf{w}) \tag{6.64}$$

where λ and $R(\mathbf{w})$ are as described in subsection 6.5.4. The task of training the network is now that of adjusting $\mathbf{w}$ such that the error,

$$\xi''(\mathbf{w}) = \mathrm{E}[\epsilon_i(\mathbf{w})] = \mathrm{E}[o_i^2] - 2\mathbf{w}^T\mathbf{p}' + \mathbf{w}^T\mathbf{R}'\mathbf{w} + \lambda R(\mathbf{w}) \tag{6.65}$$

is minimized. In equation 6.65, we have $\mathbf{p}' = \mathrm{E}[o_i\tilde{\mathbf{x}}_i]$ and $\mathbf{R}' = \mathrm{E}[\tilde{\mathbf{x}}_i\tilde{\mathbf{x}}_i^T]$. In order to minimize $\xi''(\mathbf{w})$ we use a simple gradient descent method. Starting with a randomly chosen initial weight vector $\mathbf{w}_0$, we update this vector as training examples are presented using the equation,

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha \frac{\partial \xi''(\mathbf{w}_i)}{\partial \mathbf{w}_i} \tag{6.66}$$

where α is the usual parameter specifying the step size that we wish to use. Equation 6.66 can be written as,

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha \left[-2\mathbf{p}' + 2\mathbf{R}'\mathbf{w}_i + \lambda \frac{\partial R(\mathbf{w}_i)}{\partial \mathbf{w}_i} \right]. \tag{6.67}$$

As in the standard LMS algorithm we now replace $\mathbf{p}'$ and $\mathbf{R}'$ with instantaneous estimates thereof, giving,

$$\begin{aligned} \mathbf{w}_{i+1} &= \mathbf{w}_i - \alpha \left[-2o_i\tilde{\mathbf{x}}_i + 2\tilde{\mathbf{x}}_i\tilde{\mathbf{x}}_i^T\mathbf{w}_i + \lambda \frac{\partial R(\mathbf{w}_i)}{\partial \mathbf{w}_i} \right] \\ &= \mathbf{w}_i + 2\alpha\tilde{\mathbf{x}}_i(o_i - \tilde{\mathbf{x}}_i^T\mathbf{w}_i) - \lambda' \frac{\partial R(\mathbf{w}_i)}{\partial \mathbf{w}_i} \end{aligned} \tag{6.68}$$

where $\lambda' = \alpha\lambda$.

Some initial experiments using the algorithm described by equation 6.68 have shown that it works very well when applied to the parity problem; results are described in [134]. The results obtained were significantly better than those obtained using the standard LMS algorithm, and also better than those obtained using a similar method derived by Lynch [152] which uses the standard weight decay approach. In these experiments α was chosen using a method due to Yassa [153] and λ' was set to an initial value chosen by trial and error and reduced linearly to zero during training[8]. Also, the function R used was $R(\mathbf{w}) = \sum_{i=0}^{m} \tanh(w_i^2)$. Unfortunately, we found that in tackling other problems using this algorithm it was not possible to obtain performance significantly better than that obtained using the two alternative algorithms. We suspect that better results could be obtained if it is possible to obtain a justifiable and robust method for setting the value of λ' during training, and for deciding when to stop training, as we found that when using the method described the number of non-zero weights fell quickly shortly after training commenced but later increased again. A further drawback of this algorithm is that it suffers from the usual problems associated with the LMS algorithm, which are described in full in [27].

6.9 Conclusion

In this chapter we have addressed the problem of finding an optimum structure for a Φ-net with fixed basis functions, as well as the corresponding values for the weights.

Having discussed the best way in which to define which architectures should be regarded as 'optimum', we performed an analysis of the problem based on the idea that training can be regarded as a least squares problem, and on the observation that there is often an infinite number of possible solutions to any given problem, all of which are equally good in terms of the error $\xi(\mathbf{w})$. This analysis allowed us to construct a new algorithm for finding the structure of a connectionist network, based on the important result that under certain realistic circumstances it is always possible to reduce the size of an initial network. Experimental evaluation of this algorithm, which can be used to train any of the Φ-nets described in chapter 2, shows that it usually provides a better solution than the minimum norm solution, which is often erroneously regarded as a good solution in the literature, and that it is capable of obtaining the best possible solution to a problem.

Perhaps one of the most important results presented in this chapter is the demonstration that the minimum norm weight vector is not always the optimum one when attempting to determine a good structure; this has not previously been widely appreciated. This result led us to suggest

[8]The reduction of λ' during training was suggested by Dr. M. Lynch.

a modification to the well-known weight decay training technique, although initial experiments using this idea are rather inconclusive.

Finally, we have extended our theoretical analysis of the problem of structure determination, providing results which should allow our main algorithm to be significantly improved; these results, along with further suggested improvements, provide a good basis for further research.

Chapter 7

Summary, Conclusions and Suggestions for Future Research

7.1 Summary and Conclusions

The work presented in this dissertation was originally motivated by the desire to investigate the capabilities of Φ-nets; this class of networks is similar to a class which has been available for many years, and includes many networks which have appeared in the literature. The primary reason for our interest in networks of this type was that they appeared on the basis of experimental results to be a good, but nonetheless relatively unpopular alternative to the multilayer perceptron, their main attraction being that they could often be trained significantly more quickly. Our aim was to perform a theoretical study of these networks; this was motivated by the fact that the general study of connectionist networks has often been criticized for having produced relatively little supporting theory allowing a fundamental understanding of their properties to be obtained. As our aim was to make a theoretical study, the fact that Φ-nets are quite analytically tractable provided a further motivation for the study of this particular class of networks.

We have concentrated on two particular aspects of Φ-nets: their ability to perform generalization, and the problem of how to select a good architecture on the basis of the available training examples. A subsidiary aim of the dissertation has been to compare Φ-nets with multilayer perceptrons. The generalization performance of a network is one of its most important properties, and the fact that connectionist networks have exhibited the ability to generalize is perhaps the single most important reason for the significant degree of interest that has been shown in the subject in recent years. This, and the fact that a true, fundamental understanding of generalization in the form of a widely applicable, general theory is still lacking, provided the motivation for the first part of the study. The motivation for the second part of the study was firstly that the selection of a good architecture for a network is important for good generalization and for several other reasons, and secondly that research into algorithms for the selection of a good architecture (self-structuring training algorithms) has tended to lack a good theoretical basis.

In chapter 2 we introduced and reviewed the class of Φ-nets along with the relevant supervised training algorithms, and we introduced the required material on multilayer perceptrons. In particular, we demonstrated that many standard connectionist networks, some of which are quite recent in origin and some of which have significant theoretical support, are included in the class of Φ-nets; this has not previously been widely appreciated. We also summarized some experimental results which demonstrate that Φ-nets can provide comparable performance to multilayer perceptrons. It appears that the predominance of the multilayer perceptron in connectionist network research is largely a result of unfamiliarity with the available literature, and of fashion. The main conclusion that can be drawn from this chapter is that Φ-nets can indeed be a good alternative to multilayer perceptrons in practice; we cannot, of course, claim that they will always be superior, although given the simplicity of the available training algorithms we expect that their advantage in terms of training time is likely to be present in many cases. We discussed some possible criticisms of Φ-nets, concluding that they were in most cases unlikely to cause significant problems in practice. Possible exceptions to this rule are that for some problems large weights, or a large network may be required, and this merits further research. However, in attempting to solve a given problem Φ-nets should certainly be included as candidate networks; there is certainly no reason to reject them outright.

In chapter 3 we introduced and reviewed the PAC learning formalism used in our analysis of generalization. This chapter serves mostly as a tutorial; a quite detailed exposition was deemed necessary as this formalism is relatively recent and has not been widely assimilated by the connectionist network research community. The PAC learning formalism is one of only two formalisms which have made a significant impact to date in investigating generalization. It allows us to study performance in the worst case, whereas the alternative, which is based on techniques from statistical physics, allows us to study performance in the average case. A consequence of this is that the formalism at present provides results which are very powerful, and this was one of the reasons that we chose to use it in preference to any alternative formalism. A further reason for the choice of this formalism was that results are available elsewhere for the generalization performance of feedforward networks of LTEs, providing a basis for comparison. A shortcoming of this formalism at present is that bounds obtained as a result of its use tend to be rather impractical, however significant improvements in the results are likely to be possible with further research. The growth function and the VC dimension of a network are of fundamental importance in studying the network using this formalism. Also, the concept of capacity is in general important in the study of generalization. This chapter also briefly reviewed some alternative approaches to the analysis of generalization, and some further results in computational learning theory. Finally, in an appendix related to this chapter we proved some new results showing that the hypothesis spaces corresponding to all Φ-nets, and to their l-restrictions, satisfy some conditions related to measurability which are required when using one type of PAC learning theory.

In chapter 4 we studied in detail the growth function and VC dimension associated with different types of Φ-net, obtaining several new results. We obtained tight upper bounds on both quantities which apply to completely general Φ-nets, and we provided a sufficient condition for the bound on the VC dimension to be met by a given network. We then considered lower bounds on the VC dimension for restricted versions of the most general Φ-nets. We demonstrated that the PAC learning formalism used does not allow us to directly consider the effect of using a self-structuring training algorithm, and we introduced the idea of an l-restriction in order to allow us to gain some insight into this situation. We then obtained several new results on the VC dimension of radial basis function networks having both fixed and adapting basis functions. Our results for the VC dimension of Φ-nets allow us to conclude that the common approximation, in which the VC dimension is assumed to be equal to the number of weights in a network, will often be correct in practice when considering Φ-nets, whereas for feedforward networks of LTEs, and multilayer perceptrons, it can be an underestimate. Also, unlike many VC dimension results in the literature, many of our bounds are tight. Finally, we compared our VC dimension results with various results regarding the VC dimension of feedforward networks of LTEs, and multilayer perceptrons. This comparison allowed us to suggest a theoretical explanation for the experimental observation that Φ-nets can require a relatively large number of weights in practice.

In chapter 5 we used the PAC learning formalism along with our growth function and VC dimension results to investigate the ability of Φ-nets to generalize. We derived new sufficient conditions on the number of training examples which must be learnt by a network of a given size, such that there is a specific confidence that it provides a specified performance for future inputs. These conditions apply to completely general Φ-nets. We also extended one of these conditions in an attempt to model the use of self-structuring training algorithms, by applying the idea of an l-restriction, and this led to a further new bound. Furthermore, we derived new necessary conditions for various different types of Φ-net, relating to the number of training examples which should be learnt by a network of a given size. All of our results for necessary and sufficient conditions are independent of the distribution which governs the occurrence of examples. Our necessary conditions provide quite practical bounds on the number of training examples required, however our sufficient conditions suffer from a common problem associated with this formalism: the number of training examples required is very large. The improvement of the latter bounds therefore forms an important subject for future research. These results are the only ones obtained to date specifically for Φ-nets, and they extend the work presented in [79] relating to feedforward networks of LTEs. The existence of the latter results allowed us to compare our sufficient conditions for Φ-nets with those for feedforward networks of LTEs. The results of this comparison suggest that in some circumstances (specifically, if the two networks have equal numbers of weights and can learn a specified number of training examples) Φ-nets may in general be preferable to feedforward networks of LTEs in the sense that they may require fewer training examples in order to achieve comparable generalization performance. We argued that this conclusion is likely to hold for multilayer perceptrons, as well as feedforward networks of LTEs. We performed a similar comparison of our sufficient conditions for standard training algorithms with those for self-structuring training algorithms, and obtained a similar conclusion: that the latter algorithms may in general require fewer training examples in order to provide comparable generalization performance. It is in fact probable that our current bound for self-structuring training algorithms underestimates the degree to which it may be possible to reduce the required number of training examples. Finally, a particular interpretation of one of our sufficient conditions led us to suggest that, if we wish to adapt the basis functions of a Φ-net during training, hybrid training algorithms may have specific advantages relative to standard training algorithms, and we suggested an intuitive argument in support of this view; these ideas are at present somewhat tentative and provide an interesting subject for future research.

In chapter 6 we addressed the problem of how to select the best architecture for a Φ-net as well as the values for the corresponding weights; our results apply to all Φ-nets. We argued that, given an initial network having a particular set of basis functions, a good way in which to optimize the architecture is to search for a smaller network which uses a subset of these basis functions and which is capable of providing the same error on the available sequence of training examples. An appropriate measure of the size of a Φ-net is the associated number of weights, and ideally we would like to find the smallest possible network. The obvious way in which to approach this problem is to examine networks constructed using all possible subsets of basis functions, however this is completely impractical. As Φ-nets are essentially linear when the basis functions are fixed we were able to analyse the problem of self-structuring using least squares techniques. In particular we were able to show that under certain circumstances some degree of size reduction is always possible without increasing error, and we were able to produce an exact lower bound on the extent to which this size reduction is possible. This is, to the knowledge of the author, the only result of this type obtained to date. On the basis of our theoretical analysis we designed and experimentally tested a new self-structuring training algorithm for Φ-nets, which is based on the singular value decomposition. The new algorithm was compared with the technique of calculating a minimum norm solution, and was found to provide better performance in most cases. In some cases it was able to find the best possible solution to a problem. Although the performance of the algorithm is not at present guaranteed to meet our theoretical lower bound it was found to meet or exceed it in most cases. One potential drawback of this type of self-structuring appears to be that the networks produced can require weights with relatively large magnitude. As a result of our analysis and our experimental results we were able to show that the standard weight decay technique will not necessarily find an optimum architecture. We proposed a simple modification to the weight decay technique, which we tested experimentally. These experiments were somewhat inconclusive and further research is required in order to fully evaluate this approach. Our theoretical work to this point assumed that the matrix $\mathbf{R}$ is singular, and our final step was to extend the analysis to the cases where $\mathbf{R}$ is non-singular, or where $\mathbf{R}$ is singular and has some small but non-zero singular values. This extension forms the basis for an improved version of our self-structuring training algorithm to be developed.

7.2 Suggestions for Future Research

7.2.1 Φ-Nets

We have mentioned some theoretical results which suggest that for some problems Φ-nets may require weights having very large magnitude. It is not entirely clear to what extent this is likely to be a problem in practice, although in the experimental results that we have seen it has certainly not made the use of Φ-nets impractical. A subject for future research is the investigation, both theoretical and experimental, of whether conditions exist which are likely to be encountered in practice and under which the weights required will have impractically large magnitude. Further theoretical and experimental work which compares Φ-nets with multilayer perceptrons and other networks would also be desirable, investigating in particular the conditions in which Φ-nets are particularly desirable, or otherwise, and the conditions in which Φ-nets are likely to require significantly more weights than alternative networks. Finally, theoretical work attempting to derive good sets of basis functions for Φ-nets would be useful.

7.2.2 Computational Learning Theory

There are many aspects of computational learning theory which merit further research, however we will limit our suggestions to those which are relevant to the work presented here, or which are particularly relevant to the design of connectionist networks. Perhaps the most important problem to be addressed is that the upper bounds obtained on numbers of training examples using this theory tend to be impractical, in the sense that they are large enough to be effectively inapplicable as a practical design tool. Future research on improving the current bounds is therefore important. Also, further research on versions of the formalism which make the bounds more practical by reducing the power of the theory in a sensible manner would be desirable; we have in mind further work in the spirit of Haussler *et al.* [92], in which certain assumptions have been made about the training algorithms which will be used. The modification of PAC learning theory such that self-structuring training algorithms can be fully analysed provides an important subject for future research, and similarly, it would be useful to fully analyse hybrid training algorithms using this type of formalism. Finally, a consideration of the generalization performance of Φ-nets using alternative methods, such as those based on statistical physics, or the more recent results in computational learning theory, provides obvious opportunities for further research.

7.2.3 Growth Functions and VC Dimensions

Although we have proved that $\mathcal{V}(\mathcal{F}) = W$ for various different radial basis function networks with fixed centres, and we have provided a sufficient condition for when this will be the case for general Φ-nets, a question that needs to be addressed is that of when $\mathcal{V}(\mathcal{F}) = W$ for other Φ-nets, in particular some of the other specific Φ-nets mentioned in chapter 2. Similarly, there are at present no upper bound results available for the VC dimension of radial basis function networks having adapting centres, and we suspect that our lower bound results for these networks can be improved. Also, it is possible that our VC dimension result for radial basis function networks with fixed centres and basis functions ϕ_{CUB} or ϕ_{TPS} (corollary 4.15) can be improved. Similarly, there is at present a lack of tight bounds on the VC dimension for multilayer perceptrons. Further work on the comparison of the capacity of Φ-nets with that of alternative networks using the VC dimension as well as alternative measures of capacity would be desirable.

7.2.4 Network Size and the Number of Training Examples

The most important subject for future research arising from the work of chapter 5 is the improvement of our upper bounds (sufficient conditions). One way in which to begin further research of this nature would be to attempt to improve theorems such as theorem 5.2; further improvements to our results may be obtainable using the methods suggested in subsection 5.5.2. An alternative approach to future research in this direction would be to apply to Φ-nets more recent formalisms such as that presented in [92], or the alternative theoretical approaches to generalization outlined in chapter 3. A further subject for future research is the extension of our results to networks with more than one output; this should be possible using the extensions to the PAC learning formalism which are presented in [87].

A further important subject for future research is the experimental investigation of some of the conclusions drawn from our theoretical work. In particular, experimental studies on the use of Φ-nets as opposed to multilayer perceptrons and the use of self-structuring versus standard training algorithms. Finally, our initial comments regarding hybrid training algorithms provide significant scope for experimental work, along with further theoretical work.

7.2.5 Self-Structuring Training Algorithms

The most important subject for future research arising from chapter 6 is the extension of our self-structuring training algorithm using theorem 6.12 and the extension of this theorem for the case in which $\mathbf{R}$ is non-singular, along with the experimental evaluation of the resulting algorithm. Several other possible modifications to our approach were also suggested in section 6.8, and it would also be interesting to try to produce a version of the algorithm that is capable of finding networks which do not have weights of more than a specified magnitude. Experimental comparison of our approach to self-structuring with the alternative available approaches is also an important subject for further research, and further work on the modified weight decay technique along the lines suggested in section 6.8 would be desirable. Finally, the analysis of the computational complexity of structure selection problems has to date received relatively little attention, and this provides an interesting subject for research.

Appendix A

Technical Conditions on Concept Classes and Hypothesis Spaces for Standard PAC Learning

A.1 Introduction

In our introduction to standard PAC learning in chapter 3 we mentioned some technical conditions that must be satisfied by the concept class C and the hypothesis space H; namely, that members of C and H must be Borel sets, and that C must satisfy some conditions related to measurability. In this appendix we consider these requirements in further detail. Although the requirements are not directly relevant to the research presented in chapters 4 and 5, which use *extended* PAC learning theory, an understanding of them, and in particular a knowledge of the circumstances under which they are met by a given Φ-net, is certainly useful in any consideration of Φ-nets using standard PAC learning theory. It should be remembered that these conditions are very unlikely to be violated by any system likely to be used in practice. Section A.2 considers Borel sets, and section A.3 considers conditions related to measurability.

A.2 Borel Sets

If, when using standard PAC learning, the environment is $X = \mathbb{R}^n$ for some positive integer n, we require that each $c \in C$ and each $h \in H$ is a Borel set. We will not attempt a full exposition of the properties of these sets here; a full definition is rather technical, and the reader interested in a full definition and further detail should consult Barnsley [154] or Bauer [155].

We do however make the following observations. Firstly, note that in dealing with Φ-nets having fixed basis functions it is quite often possible (and for the purposes of the requirements stated above sufficient) to think entirely in terms of hypotheses which are halfspaces in the extended space formed using the basis functions. Consequently, as halfspaces of a Euclidean space are Borel sets the relevant requirement is met in these cases.

If any exception to this rule is encountered, the mapping from input space to extended space must be such that each hypothesis is a Borel set. This requirement is not very restrictive because it is quite difficult to construct a subset of a Euclidean space which is computed by a realistic Φ-net and which is not a Borel set.

A.3 Well-Behaved Concept Classes

In our introduction to standard PAC learning in chapter 3 we mentioned that when using the environment $X = \mathbb{R}^n$ it can be necessary for the concept class C being considered to satisfy some conditions related to measurability. These conditions are encapsulated in the definition of a *well-behaved* concept class, which is given in full in Blumer *et al.* [73].

It is often (although not always) assumed that $C = H$. In this appendix we prove that the hypothesis space H computed by a Φ-net with fixed basis functions and corresponding to the class of functions $\mathcal{F}_n^{\Phi}$, and also the hypothesis space corresponding to any l-restriction of $\mathcal{F}_n^{\Phi}$, are well-behaved regardless of the set of basis functions used. This is in some sense not surprising as virtually any H likely to arise in practice in machine learning will be well-behaved, however it is a useful and important result as exceptions do exist. In order to prove the necessary results we do not use the original definition of a well-behaved H, for which [73] should be consulted for further information, but instead use the idea of a *universally separable* hypothesis space and some associated results.

Definition A.1 (Blumer *et al.* **[73])** *An hypothesis space H associated with an environment X is* universally separable *if there is a countable subset H_0 of H where each $h \in H$ is the pointwise limit of a sequence of sets $h_i \in H_0$. This means that, for all $h \in H$ there exists a sequence $h_1, h_2, \ldots$ where $h_i \in H_0$ such that for each $\mathbf{x} \in X$ there exists an n such that for all $i \geq n$, $\mathbf{x} \in h_i \iff \mathbf{x} \in h$.*

Lemma A.2 (Blumer *et al.* **[73])** *Any H which is universally separable is well-behaved.*

We can thus proceed by showing that the relevant classes of sets are universally separable. Note that not all well-behaved hypothesis spaces are universally separable. An example given in [73] is the hypothesis space defined using $X = [0, 1]$ where,

$$H = \{\{x\} \mid x \in X\}. \tag{A.1}$$

The results presented in this section were originally proved using different methods to those used below. The original proofs can be found in Holden and Rayner [109]; they are much lengthier and quite complicated, although we think that they provide greater insight into *why* hypothesis spaces computed by Φ-nets are universally separable. We have included the results in their present form, in which the first is stated as a corollary of a quite general result due to Pollard [128] and the second has a significantly shortened proof, in the interest of simplicity; as this appendix deals with rather technical conditions, which are of relatively little practical interest, we do not believe that there is anything to be gained by including lengthier and more difficult proofs of the same results.

A.3.1 Standard Φ-Nets

We first examine the case of completely standard Φ-nets. Consider the following lemma due to Pollard [128, page 38].

Lemma A.3 *Consider a finite-dimensional vector space $\mathcal{F}$ of real functions on some set S. The class of sets,*

$$\{f \geq 0 \mid f \in \mathcal{F}\} \tag{A.2}$$

is universally separable.

Recall that in chapter 4 we established that given a Φ-net $\mathcal{N} = (n, m, \Phi, \mathbf{w}, \mathcal{C})$ the class of real-valued functions,

$$\mathcal{F} = \left\{ f_{\mathbf{w}}(\mathbf{x}) = w_0 + \sum_{i=1}^{m} w_i \phi_i(\mathbf{x}) \mid \phi_i \in \Phi, \mathbf{x} \in \mathbb{R}^n \text{ and } w_i \in \mathbb{R} \text{ for } i = 0, 1, \ldots, m \right\} \qquad \text{(A.3)}$$

is a finite-dimensional vector space of functions on $\mathbb{R}^n$ (provided Φ has finite cardinality). We immediately obtain the following result.

Corollary A.4 *Let $\mathcal{N} = (n, m, \Phi, \mathbf{w}, \mathcal{C})$ be any Φ-net. Then the hypothesis space H corresponding to $\mathcal{F}_n^\Phi$ is universally separable and hence well-behaved.*

A.3.2 Φ-Nets with Self-Structuring

Having considered whether the hypothesis space computed by a standard Φ-net is well-behaved, we now ask whether the hypothesis space corresponding to the l-restriction of the class $\mathcal{F}_n^\Phi$ of functions computed by a standard Φ-net is well-behaved. In this case we cannot apply lemma A.3 directly, because it is easily verified that $\mathcal{R}^l(\mathcal{F}_n^\Phi)$ is not necessarily a vector space (if $f_1 \in \mathcal{R}^l(\mathcal{F}_n^\Phi)$, $f_2 \in \mathcal{R}^l(\mathcal{F}_n^\Phi)$ and $f_1 \neq f_2$ it is not always true that $f_1 + f_2 \in \mathcal{R}^l(\mathcal{F}_n^\Phi)$). However, recall that in chapter 4 we noted that $\mathcal{R}^l(\mathcal{F}_n^\Phi)$ can be expressed as,

$$\mathcal{R}^l(\mathcal{F}_n^\Phi) = \bigcup_{i=1}^{N(l,W)} \mathcal{F}_n^{\Phi_i} = \mathcal{F}_n^{\Phi_1} \cup \mathcal{F}_n^{\Phi_2} \cup \cdots \cup \mathcal{F}_n^{\Phi_{N(l,W)}} \qquad \text{(A.4)}$$

where each $\mathcal{F}_n^{\Phi_i}$ is the class of functions computed by a standard Φ-net. Let H be the hypothesis space corresponding to $\mathcal{R}^l(\mathcal{F}_n^\Phi)$. From equation A.4 we can express H as,

$$H = \bigcup_{i=1}^{N(l,W)} H_i \qquad \text{(A.5)}$$

where H_i is the hypothesis space corresponding to $\mathcal{F}_n^{\Phi_i}$. As each H_i is therefore the hypothesis space computed by a standard Φ-net it is universally separable by corollary A.4 and consequently we have the following result.

Corollary A.5 *Let H be the hypothesis space corresponding to the l-restriction $\mathcal{R}^l(\mathcal{F}_n^\Phi)$ of $\mathcal{F}_n^\Phi$ where $\mathcal{F}_n^\Phi$ is the class of functions computed by any Φ-net having $W \geq 3$ weights, $l \geq 2$ and $W > l$. Then H is universally separable and hence well-behaved.*

Proof This follows directly from equation A.5 and definition A.1. Because each H_i is universally separable it has an associated countable subset $H_0^{(i)}$ such that each $h \in H_i$ is the pointwise limit of a sequence of sets in $H_0^{(i)}$. We therefore define the countable subset H_0 of H as,

$$H_0 = \bigcup_{i=1}^{N(l,W)} H_0^{(i)} \qquad \text{(A.6)}$$

and note that because H can be written as shown in equation A.5, each $h \in H$ is the pointwise limit of a sequence of sets $h_i \in H_0$ and consequently H is universally separable. □

A.4 Conclusion

In this appendix we have considered some technical conditions on concepts classes and hypothesis spaces which are required when using standard PAC learning theory. We have argued that the requirement that concepts and hypotheses are Borel sets will present no problem in practice, especially when dealing with Φ-nets having fixed basis functions, and we have proved that hypothesis spaces computed by Φ-nets with and without self-structuring (where Φ-nets with self-structuring are modelled using an l-restriction) are always well-behaved as defined in [73].

Bibliography

[1] Stephen M. Omohundro. Efficient algorithms with neural network behavior. *Complex Systems*, 1:273–347, 1987.

[2] B. D. Ripley. Statistical aspects of neural networks. To appear in the Proceedings of Séminaire Européen de Statistique, Sandbjerg, Denmark, June 1992.

[3] Marvin L. Minsky and Seymour A. Papert. *Perceptrons. An Introduction to Computational Geometry. Expanded Edition.* MIT Press, 1988.

[4] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, 1973.

[5] Richard P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, pages 4–22, April 1987.

[6] Richard P. Lippmann. Pattern classification using neural networks. *IEEE Communications Magazine*, pages 47–64, November 1989.

[7] Den R. Hush and Bill G. Horne. Progress in supervised neural networks: What's new since Lippmann? *IEEE Signal Processing Magazine*, pages 8–39, January 1993.

[8] F. Girosi and T. Poggio. Networks for learning. A view from the theory of approximation of functions. In Paolo Antognetti and Veljko Milectinović, editors, *Neural Networks. Concepts, Applications and Implementations*, pages 110–154. Prentice Hall, 1991. Volume 1.

[9] A. G. Ivakhnenko. Polynomial theory of complex systems. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-1(4):364–378, October 1971.

[10] Andrew R. Barron and Roger L. Barron. Statistical learning networks: A unifying view. In *1988 Symposium on the Interface: Statistics and Computer Science*, pages 192–203, April 1988.

[11] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Workshop on Computational Learning Theory*, pages 144–152, 1992.

[12] S. Renals and R. Rohwer. Phoneme classification experiments using radial basis functions. In *Proceedings of the International Joint Conference on Neural Networks*, pages I–461–I–467, June 1989.

[13] P. C. Bressloff and D. J. Weir. Neural networks. *GEC Journal of Research*, 8(3):151–169, 1991.

[14] Yoh-Han Pao. *Adaptive Pattern Recognition and Neural Networks.* Addison Wesley, 1989.

[15] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the Theory of Neural Computation.* Santa Fe Institute Studies in the Sciences of Complexity. Addison-Wesley, 1991.

[16] D. Lowe and A. R. Webb. Time series prediction by adaptive networks: A dynamical systems perspective. *IEE Proc. Pt. F - Radar and Signal Processing*, 138(1):17–24, February 1991.

[17] Thomas M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, EC-14:326–334, 1965.

[18] Nils J. Nilsson. *Learning Machines. Foundations of Trainable Pattern-Classifying Systems.* McGraw-Hill, 1965.

[19] David E. Rumelhart, James L. McClelland, and The PDP Research Group. *Parallel Distributed Processing. Explorations in the Microstructure of Cognition*, volume 1. MIT Press, 1986.

[20] Michael R. Garey and David S. Johnson. *Computers and Intractability. A guide to the theory of NP-Completeness.* Freeman, 1979.

[21] Stephen Judd. On the complexity of loading shallow neural networks. *Journal of Complexity*, 4(3):177–192, September 1988.

[22] J. Stephen Judd. *Neural Network Design and the Complexity of Learning.* MIT Press, 1990.

[23] Avrim L. Blum and Ronald L. Rivest. Training a 3-node neural network is NP-complete. *Neural Networks*, 5:117–127, 1992.

[24] John F. Kolen and Ashok K. Goel. Learning in parallel distributed processing networks: Computational complexity and information content. *IEEE Transactions on Systems, Man and Cybernetics*, 21(2):359–367, March/April 1991.

[25] Pekka Orponen. Neural networks and complexity theory. In I. M. Havel and V. Koubek, editors, *Proceedings of the 17th International Symposium on the Mathematical Foundations of Computer Science*, pages 50–61, August 1992. Lecture Notes in Computer Science, 629. Published by Springer-Verlag.

[26] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C.* Cambridge University Press, second edition, 1992.

[27] Simon Haykin. *Adaptive Filter Theory.* Prentice-Hall, second edition, 1991.

[28] M. R. Lynch and P. J. W. Rayner. The properties and implementation of the non-linear vector space connectionist model. In *Proceedings of the IEE 1st International Conference on Artificial Neural Networks*, October 1989.

[29] David Lowe. Adaptive radial basis function nonlinearities and the problem of generalisation. In *Proceedings of the First IEE International Conference on Artificial Neural Networks*, pages 171–175, 1989.

[30] John Moody and Christian J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1989.

[31] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[32] Martin Casdagli. Nonlinear prediction of chaotic time series. *Physica D. Nonlinear Phenomena*, 35:335–356, 1989.

[33] S. Chen, G. J. Gibson, and C. F. N. Cowan. Adaptive channel equalisation using a polynomial-perceptron structure. *IEE Proc. Pt. I - Communications, Speech and Vision*, 137(5):257–264, October 1990.

[34] A. Lapedes and R. Farber. Nonlinear signal processing using neural networks: Prediction and system modelling. Technical Report LA-UR-87-2662, Los Alamos National Laboratory, July 1987.

[35] Donald F. Specht. Generation of polynomial discriminant functions for pattern recognition. *IEEE Transactions on Electronic Computers*, EC-16(3):308–319, June 1967.

[36] P. J. W. Rayner and M. R. Lynch. Complexity reduction in Volterra connectionist modelling by consideration of output mapping. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing 1990*, 1990.

[37] C. Lee Giles and Tom Maxwell. Learning, invariance and generalization in high-order neural networks. *Applied Optics*, 26(23):4972–4978, December 1987.

[38] Tom Maxwell, C Lee Giles, and YC Lee. Generalization in neural networks: The contiguity problem. In *IEEE First International Conference on Neural Networks*, volume 2, June 1987.

[39] D. S. Broomhead and David Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[40] David H. Wolpert. Constructing a generalizer superior to NETtalk via a mathematical theory of generalization. *Neural Networks*, 3:445–452, 1990.

[41] M. J. D. Powell. Radial basis functions for multivariable interpolation: A review. Technical Report #DAMTP 1985/NA12, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, October 1985.

[42] M. J. D. Powell. The theory of radial basis function approximation in 1990. Technical Report #DAMTP 1990/NA11, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, December 1990.

[43] Charles A. Micchelli. Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constructive Approximation*, 2:11–22, 1986.

[44] Tomaso Poggio and Federico Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, September 1990.

[45] J. Park and I. W. Sandberg. Universal approximation using radial-basis-function networks. *Neural Computation*, 3:246–257, 1991.

[46] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

[47] Eric J. Hartman, James D. Keeler, and Jacek M. Kowalski. Layered neural networks with Gaussian hidden units as universal approximations. *Neural Computation*, 2:210–215, 1990.

[48] S. Chen, S. A. Billings, and P. M. Grant. Recursive hybrid algorithm for non-linear system identification using radial basis function networks. *International Journal of Control*, 55(5):1051–1070, 1992.

[49] M. T. Musavi, W. Ahmed, K. H. Chan, K. B. Faris, and D. M. Hummels. On the training of radial basis function classifiers. *Neural Networks*, 5(4):595–603, July-August 1992.

[50] T. Poggio and F. Girosi. Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, 247:978–982, February 1990.

[51] Tomaso Poggio and Federico Girosi. A theory of networks for approximation and learning. Technical Report A.I. Memo 1140, C.B.I.P. Paper 31, Massachusetts Institute of Technology and Center for Biological Information Processing Whitaker College, July 1989.

[52] Tomaso Poggio and Federico Girosi. Extensions of a theory of networks for approximation and learning: dimensionality reduction and clustering. Technical Report A.I. Memo 1167, Massachusetts Institute of Technology and Center for Biological Information Processing Whitaker College, April 1990.

[53] Federico Girosi, Tomaso Poggio, and Bruno Caprile. Extensions of a theory of networks for approximation and learning: outliers and negative examples. Technical Report A.I. Memo 1220, C.B.I.P. Paper 46, Massachusetts Institute of Technology and Center for Biological Information Processing Whitaker College, July 1990.

[54] G. de Barra. *Measure Theory and Integration.* Ellis Horwood Limited, 1981.

[55] R. W. Prager and F. Fallside. The modified Kanerva model for automatic speech recognition. *Computer Speech and Language*, 3:61–81, 1989.

[56] Pentti Kanerva. *Self-Propagating Search: A Unified Theory of Memory.* PhD thesis, Stanford University, Department of Philosophy, February 1984.

[57] T. J. W. Clarke, R. W. Prager, and F. Fallside. The modified Kanerva model: Theory and results for real-time word recognition. *IEE Proc. Pt. F - Radar and Signal Processing*, 138(1):25–31, February 1991.

[58] Richard W. Prager. Some experiments with fixed non-linear mappings and single layer networks. Technical Report CUED/F-INFENG/TR.106, Cambridge University Engineering Department, July 1992.

[59] J. S. Albus. A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Transactions of the ASME*, pages 220–227, 1975.

[60] W. Thomas Miller, III, Filson H. Glanz, and L. Gordon Kraft, III. CMAC: An associative neural network alternative to backpropagation. *Proceedings of the IEEE*, 78(10):1561–1567, October 1990.

[61] Stephen I. Gallant and Donald Smith. Random cells: An idea whose time has come and gone ... and come again? In *IEEE First International Conference on Neural Networks*, volume 2, pages 671–678, 1987.

[62] P. J. W. Rayner and M. R. Lynch. A new connectionist model based on a non-linear adaptive filter. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, May 1989.

[63] M. R. Lynch and P. J. W. Rayner. Optical character recognition using a new connectionist model. In *Proceedings of the IEE International Conference on Image Processing*, July 1989.

[64] U. Kreßel, J. Franke, and J. Schurmann. Polynomial classifier versus multilayer perceptron. Unpublished manuscript. Daimler-Benz AG, Reseach Center Ulm, Wilhelm-Runge-Str. 11, 7900 Ulm, F.R.Germany.

[65] J. J. Rajan and P. J. W. Rayner. Time series classification using the Volterra connectionist model and Bayes decision theory. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 1, pages I–601–I–604, April 1993.

[66] M. Niranjan and F. Fallside. Neural networks and radial basis functions in classifying static speech patterns. *Computer Speech and Language*, 4(3):275–289, July 1990. Also available as Cambridge University Engineering Department Report number CUED/F-INFENG/TR 22 (1988).

[67] Yuchun Lee. Handwritten digit recognition using k nearest-neighbor, radial-basis function, and backpropagation neural networks. *Neural Computation*, 3(3):440–449, 1991.

[68] Saburo Muroga. *Threshold Logic and its Applications.* Wiley-Interscience, 1971.

[69] Michael E. Saks. Slicing the hypercube. In Keith Walker, editor, *Surveys in Combinatorics, 1993*, pages 211–255. Cambridge University Press, 1993.

[70] V. N. Vapnik and A. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[71] Martin Anthony and Norman Biggs. *Computational Learning Theory.* Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1992.

[72] N. Sauer. On the density of families of sets. *Journal of Combinatorial Theory (A)*, 13:145–147, 1972.

[73] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association for Computing Machinery*, 36(4):929–965, October 1989.

[74] M. Anthony and J. Shawe-Taylor. A result of Vapnik with applications. Technical Report LSE-MPS-18, London School of Economics, November 1991. To appear in Discrete Applied Mathematics.

[75] Lorenza Saitta and Francesco Bergadano. Pattern recognition and Valiant's learning framework. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(2):145–155, February 1993.

[76] Yaser Abu-Mostafa. The Vapnik-Chervonenkis dimension: Information versus complexity in learning. *Neural Computation*, 1:312–317, 1989.

[77] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[78] Vladimir Vapnik. *Estimation of Dependences Based on Empirical Data.* Springer-Verlag, 1982.

[79] Eric B. Baum and David Haussler. What size net gives valid generalization? *Neural Computation*, 1:151–160, 1989.

[80] Balas K. Natarajan. *Machine Learning: A Theoretical Approach.* Morgan Kaufmann Publishers INC, 1991.

[81] Dana Angluin. Computational learning theory: Survey and selected bibliography. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on the Theory of Computing*, pages 351–369, May 1992.

[82] Michael J. Kearns. *The Computational Complexity of Machine Learning.* ACM-MIT Distinguished Dissertation Series. The MIT Press, 1990.

[83] David Haussler. Probably approximately correct learning. Technical Report UCSC-CRL-90-16, University of California at Santa Cruz, May 1990.

[84] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Classifying learnable geometric concepts with the Vapnik-Chervonenkis dimension. In *18th ACM Symposium on the Theory of Computing*, pages 273–282, 1986.

[85] Andrzej Ehrenfeucht, David Haussler, Michael Kearns, and Leslie Valiant. A general lower bound on the number of examples needed for learning. *Information and Computation*, 82:247–261, 1989.

[86] Martin Anthony and Norman Biggs. Computational learning theory for artificial neural networks. Technical Report LSE-MPS-44, London School of Economics, February 1993. To appear in "Mathematical Studies of Neural Networks" (ed. J. G. Taylor), Elsevier, 1993.

[87] John Shawe-Taylor and Martin Anthony. Sample sizes for multiple-output threshold networks. *Network*, 2:107–117, 1991.

[88] D. Haussler. Generalizing the PAC model: Sample size bounds from metric dimension-based uniform convergence results. In *Proc 30th IEEE Symposium on the Foundations of Computer Science*, pages 40–45, 1989.

[89] David Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation*, 100(1):78–150, September 1992.

[90] Michael J. Kearns and Robert E. Schapire. Efficient distribution-free learning of probabilistic concepts. In *IEEE 31st Annual Symposium on Foundations of Computer Science*, volume 1, pages 382–391, 1990.

[91] Paul Fischer, Stefan Polt, and Hans Ulrich Simon. Probably almost Bayes decisions. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 88–94, 1991.

[92] David Haussler, Michael Kearns, and Robert Schapire. Bounds on the sample complexity of Bayesian learning using information theory and the VC dimension. In *Proceedings of the Fourth Workshop on Computational Learning Theory (COLT)*, 1991.

[93] H. S. Seung, H. Sompolinsky, and N. Tishby. Statistical mechanics of learning from examples. *Physical Review A*, 45(8):6056–6091, April 1992.

[94] F. Kanaya and S. Miyake. Bayes statistical behavior and valid generalization of pattern classifying neural networks. In *Proceedings COGNITIVA90*, pages 35–44, November 1990.

[95] Fumio Kanaya and Shigeki Miyake. Bayes statistical behavior and valid generalization of pattern classifying neural networks. *IEEE Transactions on Neural Networks*, 2(4):471–475, July 1991.

[96] V. V. Anshelevich, B. R. Amirikian, A. V. Lukashin, and M. D. Frank-Kamenetskii. On the ability of neural networks to perform generalization by induction. *Biological Cybernetics*, 61:125–128, 1989.

[97] Eric A. Wan. Neural network classification: A Bayesian interpretation. *IEEE Transactions on Neural Networks*, 1(4):303–305, December 1990.

[98] John Denker, Daniel Schwartz, Ben Wittner, Sara Solla, Richard Howard, Lawrence Jackel, and John Hopfield. Large automatic learning, rule extraction and generalization. *Complex Systems*, 1:877–922, 1987.

[99] David H. Wolpert. A mathematical theory of generalization: Part I. *Complex Systems*, 4:151–200, 1990.

[100] David H. Wolpert. A mathematical theory of generalization: Part II. *Complex Systems*, 4:201–249, 1990.

[101] Luc Devroye. Automatic pattern recognition: A study of the probability of error. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(4):530–543, July 1988.

[102] David J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, May 1992.

[103] David J. C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, May 1992.

[104] Yann le Cun. Generalization and network design strategies. In R. Pfeifer, Z. Schreter, F. Fogelman-Soulié, and L. Steels, editors, *Connectionism in Perspective*, pages 143–155. Elsevier Science Publishers B.V. (North-Holland), 1989.

[105] Jocelyn Sietsma and Robert J. F. Dow. Creating artificial neural networks that generalize. *Neural Networks*, 4:67–79, 1991.

[106] I. Guyon, V. Vapnik, B. Boser, L. Botton, and S. A. Solla. Structural risk minimization for character recognition. In *Proceedings of NIPS 1991*, 1991.

[107] Bradley Efron. Computers and the theory of statistics: Thinking the unthinkable. *SIAM Review*, 21(4):460–480, October 1979.

[108] Sean B. Holden and Peter J. W. Rayner. Generalization and learning in Volterra and radial basis function networks: A theoretical analysis. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages II–273–II–276, 1992.

[109] Sean B. Holden and Peter J. W. Rayner. Generalization and PAC learning: Some new results for the class of generalized single layer networks. *IEEE Transactions on Neural Networks*, 1993. Accepted for publication. To appear in a future issue. A preliminary version of this article was published as Cambridge University Engineering Department Report number CUED/F-INFENG/TR86, May 1992.

[110] Sean B. Holden and Martin Anthony. Quantifying generalization in linearly weighted neural networks. Technical Report CUED/F-INFENG/TR.113 1993, Cambridge University Engineering Department, 1993. Submitted to Complex Systems.

[111] Sean B. Holden. Neural networks and the VC dimension. Technical Report CUED/F-INFENG/TR.119 (1992), Cambridge University Engineering Department, December 1992. To appear in the Proceedings of the IMA International Conference on Mathematics in Signal Processing, Warwick, 1992.

[112] Sean B. Holden. Valid generalization in radial basis functions and modified Kanerva models. In *Proceedings of the Third IEE International Conference on Artificial Neural Networks*, pages 100–104, May 1993. Conference publication number 372.

[113] Martin Anthony and Sean B. Holden. On the power of linearly weighted neural networks. To appear in the Proceedings of the International Conference on Neural Networks (ICANN) 1993, 1993.

[114] Martin Anthony and Sean B. Holden. On the power of polynomial discriminators and radial basis function networks. In *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory*, pages 158–164, July 1993.

[115] R. S. Wenocur and R. M. Dudley. Some special Vapnik-Chervonenkis classes. *Discrete Mathematics*, 33:313–318, 1981.

[116] R. M. Dudley. Central limit theorems for empirical measures. *Annals of Probability*, 6:899–929, 1978.

[117] J. A. Green. *Sets and Groups. A First Course in Algebra.* Routledge & Kegan Paul, 1988.

[118] S. Young and T. Downs. Generalisation in higher order neural networks. *Electronics Letters*, 29(16):1491–1493, August 1993.

[119] Norman L. Biggs. *Discrete Mathematics.* Oxford University Press, 1985.

[120] Nira Dyn and Charles A. Micchelli. Interpolation by sums of radial functions. *Numerische Mathematik*, 58:1–9, 1990.

[121] Wolfgang Maass. Bounds for the computational power and learning complexity of analog neural nets. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 335–344, 1993.

[122] Wolfgang Maass. Neural nets with superlinear VC-dimension. Unpublished Manuscript. Institute for Theoretical Computer Science, Technische Universitaet Graz, Klosterwiesgasse 32/2, A-8010 Graz, Austria, 1993.

[123] Eric B. Baum. On the capabilities of multilayer perceptrons. *Journal of Complexity*, 4(3):193–215, September 1988.

[124] Peter L. Bartlett. Lower bounds on the Vapnik-Chervonenkis dimension of multi-layer threshold nets. Technical Report IML92/3, University of Queensland, Department of Electrical Engineering, Intelligent Machines Laboratory, September 1992.

[125] Eduardo D. Sontag. Feedforward nets for interpolation and classification. *Journal of Computer and System Sciences*, 45(1):20–48, August 1992.

[126] Marco Budinich and Eduardo Milotti. Properties of feedforward neural networks. *Journal of Physics A: Mathematical and General*, 25:1903–1914, 1992.

[127] Eduardo D. Sontag. Some topics in neural networks and control. Technical Report LS93-02, Department of Mathematics, Rutgers University, New Brunswick, NJ 08903, July 1993.

[128] David Pollard. *Convergence of Stochastic Processes.* Springer-Verlag, 1984.

[129] David Haussler. Private communication.

[130] Martin Anthony. Private communication.

[131] Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, August 1991.

[132] David Cohn and Gerald Tesauro. How tight are the Vapnik-Chervonenkis bounds? *Neural Computation*, 4(2):249–269, March 1992.

[133] M. R. Lynch, P. J. Rayner, and S. B. Holden. Removal of degeneracy in adaptive Volterra networks by dynamic structuring. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing 1991*, pages 2069–2072, 1991.

[134] M. R. Lynch, S. B. Holden, and P. J. Rayner. Complexity reduction in Volterra connectionist networks using a self-structuring LMS algorithm. In *Proceedings of the IEE Second International Conference on Artificial Neural Networks*, pages 44–48, 1991.

[135] Jyh-Han Lin and Jeffrey Scott Vitter. Complexity results on learning by neural nets. *Machine Learning*, 6:211–230, 1991.

[136] Robert Michael Debenham. *Studies in Learning and Feature Location with an Artificial Neural Network.* PhD thesis, Cambridge University Engineering Department, December 1991.

[137] Visakan Kadirkamanathan. *Sequential Learning in Artificial Neural Networks.* PhD thesis, Cambridge University Engineering Department, October 1991.

[138] Yann Le Cun, John S. Denker, and Sara A. Solla. Optimal brain damage. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan Kaufmann, 1990.

[139] Ethem Alpaydin. GAL: Networks that grow when they learn and shrink when they forget. Technical Report TR-91-032, International Computer Science Institute, 1947 Center Street, Suite 600, Berkeley, California 94704-1105, May 1991.

[140] Dale E. Nelson and Steven K. Rogers. A taxonomy of neural network optimality. In *Proceedings of the National Aerospace Electronics Conference (NAECON) 1992*, volume 3, pages 894–899, May 1992.

[141] Lennart Ljung. *System Identification: Theory for the User.* Prentice-Hall, 1987.

[142] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* Johns Hopkins, second edition, 1989.

[143] Gilbert Strang. *Introduction to Applied Mathematics.* Wellesley-Cambridge Press, 1986.

[144] Virginia C. Klema and Alan J. Laub. The singular value decomposition: Its computation and some applications. *IEEE Transactions on Automatic Control*, AC-25(2):164–176, April 1980.

[145] The Math Works Inc. *Matlab Reference Guide*, August 1992.

[146] Steven J. Nowlan and Geoffrey E. Hinton. Simplifying neural networks by soft weight-sharing. Technical Report CRG-TR-91-4, Connectionist Research Group, University of Toronto, October 1991.

[147] James R. Bunch and Christopher P. Nielsen. Updating the singular value decomposition. *Numerische Mathematik*, 31:111–129, 1978.

[148] Marc Moonen. *Jacobi-Type Updating Algorithms for Signal Processing, Systems Identification and Control.* PhD thesis, Katholieke Universiteit Leuven, Faculteit Toegepaste Wetenschappen, Departement Elektrotechniek, Afdeling Esat, K. Mercierlaan 94 — 3001 Heverlee (Belgium), November 1990.

[149] S. Chen, S. A. Billings, and W. Luo. Orthogonal least squares methods and their application to non-linear system identification. *International Journal of Control*, 50(5):1873–1896, 1989.

[150] S. Chen, C. F. N. Cowan, and P. M. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, 2(2):302–309, March 1991.

[151] A. S. Young. The bivar criterion for selecting regressors. *Technometrics*, 24(3):181–189, August 1982.

[152] M. R. Lynch. *Adaptive Techniques in Signal Processing and Connectionist Models.* PhD thesis, Cambridge University Engineering Department, June 1990.

[153] Fathy F. Yassa. Optimality in the choice of the convergence factor for gradient based adaptive algorithms. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-35(1):48–59, January 1987.

[154] Michael Barnsley. *Fractals Everywhere.* Academic Press, 1988.

[155] Heinz Bauer. *Probability Theory and Elements of Measure Theory.* Academic Press, 1981.