Designing and Implementing Embedded Fault Diagnosis Systems Using MLP and RBFN Classifiers

Thesis submitted for the degree of
Doctor of Philosophy
at the University of Leicester

by

Yuhua Li
BEng, MEng

Department of Engineering
University of Leicester

2003

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Designing and Implementing Embedded Fault Diagnosis Systems Using MLP and RBFN Classifiers

by

Yuhua Li 李毓华
BEng, MEng

Declaration of Originality

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy in the Department of Engineering, The University of Leicester, UK. All work recorded in this thesis is original unless otherwise acknowledged in the thesis or by reference. No part of it has been submitted for any degree, either to the University of Leicester or to any other university.

Yuhua Li
2003

To JunYan and WenTing

Acknowledgements

I owe a debt of gratitude to my supervisor, Dr Michael Pont, who has consistently supported and encouraged me throughout the duration of the work. I am most grateful for his patience and tolerance with my English.

I would like to gratefully acknowledge the financial support of an ORS award from CVCP, a scholarship from the University of Leicester, and an additional scholarship raised by Dr Pont during the course of this research work.

I owe a debt of thanks to Dr John Twiddle for the supply of simulation data for an engine cooling system, and experimental data for an engine aspiration system. I am also deeply grateful for his patience in checking the English in my papers and part of this thesis.

Chinmay Parikh deserves special thanks, for the co-operation of research work and paper writing. I am indebted to my other colleagues: Royan Ong, (Jason) Chang Liang Lim, Ken Boon Goh, Aamer I. Bhatti, Yu Xu, Yuehe Wang, Chen Pang Wong and Daljeet Gill, for making the BTSP group an interesting and enjoyable place to work.

Designing and Implementing Embedded Fault Diagnosis Systems Using MLP and RBFN Classifiers

Yuhua Li (BEng, MEng)
Department of Engineering
University of Leicester

李毓华
PhD thesis
2003

Abstract

As new microcontrollers and related processors have become available, it has become possible to create embedded systems for condition monitoring and fault diagnosis (CMFD). This thesis explores how two popular classifiers: the MultiLayer Perceptron (MLP) and Radial Basis Function neural Network (RBFN), can be most effectively employed in embedded CMFD systems.

The design of an embedded CMFD system can be considered to consist of three stages, involving data pre-processing, classification, and post-processing of classifier outputs. The thesis makes contributions to each of these phases as follows.

First, the thesis describes a novel separability analysis method which is able to predict the relative effectiveness of pre-processing techniques. An important aspect of this method is that the separability is derived from a non-parametric analysis: it therefore requires no assumptions to be made about the underlying distribution of the data.

Second, a design methodology is derived that may be used to help the software engineer select between the use of MLP or RBFN classifiers in the CMFD system, depending on the particular system requirements. The design methodology is the result of a comprehensive series of empirical studies. The comparison criteria used are those of particular relevance in embedded CMFD applications. These include classification performance in the presence of unknown faults, with multiple faults, and with limited training data. The criteria also include processor and memory requirements.

Third, the thesis develops a novel technique that allows the user to determine an appropriate threshold for interpreting the outputs of a trained RBFN classifier. Results from two experiments demonstrate that this technique can be used to improve the performance of RBFN classifiers in practical CMFD applications where 'unknown faults' may occur.

Designing and Implementing Embedded Fault Diagnosis Systems Using MLP and RBFN Classifiers

Yuhua Li
Department of Engineering, University of Leicester

List of Publications

A number of papers have been published during the course of the work described in this thesis. They are listed below.

Directly-related journal papers

Li, Y., Pont, M.J., Jones, N.B. "Improving the performance of radial basis function classifiers in condition monitoring and fault diagnosis applications where 'unknown' faults may occur," Pattern Recognition Letters, vol.23, no.5, 2002, pp569-577.

Li, Y., Pont, M.J. "On selecting pre-processing techniques for fault classification using neural networks," International Journal of Knowledge-Based Intelligent Engineering Systems, vol.6, no.2, 2002, pp80-87.

Li, Y., Pont, M.J., Jones, N.B. "Using MLP and RBF classifiers in embedded condition monitoring and fault diagnosis applications," Transactions of the Institute of Measurement and Control, vol. 23, no.5, 2001, pp315-343

Jones, N.B., Li, Y. "A review on condition monitoring and fault diagnosis for diesel engines," Tribotest Journal, vol.6, no.3, 2000, pp267-291.

Directly-related conference papers

Li, Y., Pont, M.J., Parikh, C.R., Jones, N.B. "Comparing the Performance of Three Neural Classifiers for Use in Embedded Applications," Soft Computing Techniques and Applications. Physica-Verlag: Heidelberg, New York, 2000, pp34-39. ISBN 3-7908-1257-9.

Li, Y., Pont, M.J., Parikh, C.R., Jones, N.B. "Using a Combination of RBFN, MLP and kNN Classifiers for Engine Misfire Detection," Soft Computing Techniques and Applications. Physica-Verlag: Heidelberg, New York, 2000, pp46-51. ISBN 3-7908-1257-9.

Li, Y., Pont, M. J., Jones, N. B. "A comparison of the performance of radial basis function and multi-layer Perceptron networks in condition monitoring and fault diagnosis," Proceedings of the International conference on condition monitoring, Swansea, UK, 12th-15th, April 1999, pp577-592.

Li, Y., Jones, N. B., Pont, M. J. "Applying neural networks and fuzzy logic to fault diagnosis: a review," Proceedings of recent advances in soft computing'98, Leicester, UK, July 1998, pp104-119.

Associated publications

Li, Y., Zheng, H., Jones, N.B., Pont, M.J. "Multi-channel rotational speed measurement: a software based approach," Measurement and Control, vol. 31, no. 8, 1998, pp229-231.

Parikh, C.R. Pont, M.J. Li, Y., Jones, N.B. "Investigating the performance of MLP classifiers where limited training data are available for some classes," Soft Computing Techniques and Applications. Physica-Verlag: Heidelberg, New York, 2000, pp22-27. ISBN 3-7908-1257-9.

Parikh, C.R., Pont M.J., Li, Y., Jones, N.B. "Neural networks for condition monitoring and fault diagnosis: the effect of training data on classifier performance," Proceedings of the International conference on condition monitoring, Swansea, UK, April 1999, pp237-244.

Parikh, C.R., Pont, M.J., Li, Y., Jones, N.B., Twiddle, J.A. "Towards a flexible application framework for data fusion using real-time design patterns," EUFIT'98, Germany, Sep. 1998, pp1131-1135.

Pont, M.J., Ong, R.H.L., Parikh, C.R., Kureemun, R., Wong, C.P., Peasgood, W., Li, Y. "A selected of pattern for reliable embedded systems," Fourth European Conference on Pattern Languages of Programming and Computing (EuroPlop'99), July 1999, Bad Irsee, Germany. Available from "http://www.argo.be/europlop/writers.htm"

Pont, M.J., Parikh, C.R., Li, Y., Wong, C.P. "The design of embedded system using software pattern," Proceedings of the International conference on condition monitoring, Swansea, UK, April 1999, pp221-236.

Abbreviations

CMFD:	Condition Monitoring and Fault Diagnosis
MLP:	Multi-Layer Perceptrons
RBFN:	Radial Basis Function Networks
CPU:	Central Processing Unit
ROM:	Read Only Memory
RAM:	Random Access Memory
MM:	Mathematical Model for static fault diagnosis
DC:	Diesel engine Cooling system model
BC:	Breast Cancer diagnosis
OBD II:	On-Board Diagnosis generation 2
VC dimension:	Vapnik-Chervonenkis dimension

1
DESIGNING EMBEDDED CMFD SYSTEMS

1.1 Introduction

Embedded condition monitoring and fault diagnosis (CMFD) components are becoming an increasingly important feature of complex modern systems. This is due to the growing demands for plant availability, reliability, maintainability, safety, quality and cost efficiency (Patton, *et al*, 1989; Chan, *et al*, 1997; Flammini, *et al*, 2001; Min, *et al*, 2002). In this context, a fault represents an unpermitted deviation of at least one characteristic property or parameter of the plant from the 'acceptable', 'usual' or 'standard' condition (Isermann & Balle, 1997; Isermann, 1997). The objective of CMFD is then to determine if such a fault is present and - if so - to identify the size, location and time of occurrence (Sick, 2002).

The aim of the programme of work described in this thesis was to investigate how Multi-Layer Perceptrons (MLPs) and Radial Basis Function neural Networks (RBFNs) techniques could be most effectively applied in embedded CMFD applications. The particular focus of this work was on what are referred to here as "three-stage neural classifiers". These classifiers involve the pre-processing of

raw data from the plant, the design of suitable (neural) classifiers, and the post-processing of classifier outputs. In such a three-stage system, each stage contributes significantly to the system performance (Theodoridis & Koutroumbas, 1999).

This introductory chapter begins by explaining the importance of CMFD using neural networks (Section 1.2), before discussing the distinctive features of the classifiers used for CMFD purposes (Section 1.3). In Section 1.4 it is argued that the development of an embedded implementation of a classifier imposes severe restrictions in memory and CPU power. Section 1.5 then describes the process of designing CMFD classification systems using neural networks. Following this introduction, Section 1.6 explains the scope of the project described in this thesis, and finally Section 1.7 provides an overview of the thesis itself.

1.2 CMFD using neural networks

A variety of CMFD methods have been developed in the last two decades (Isermann & Balle, 1997; Chantler, *et al*, 1998; Sick, 2002). In the early years, research efforts concentrated on the development of methods for linear dynamic plants (Willsky, 1976), while in the 1980s, the model-based approach formed the core of the CMFD methodology (Frank & KoppenSeliger, 1997; Isermann, 1984; Gertler, 1993; Patton, *et al*, 1995). More recent research has mainly focused on the development of CMFD methods suitable for use in non-linear systems with uncertainty (Frank, 1994; Gertler, *et al*, 1995; Patton & Chen, 1997; Le Riche, *et al*, 2001).

Because of the increasing complexity of modern plants and the developments in artificial intelligence (Marks, 1993), intelligence-based CMFD methods were a particular focus of research during the 1990s (Patton & Lopez-Toribio, 1998; Ayoubi & Isermann, 1997; Frank & KoppenSeliger, 1997). In particular, artificial neural networks have been extensively applied in practical CMFD systems (Kramer & Leonard, 1990; Li, *et al*, 1998; Grimmelius, *et al*, 1999). Examples include: engine diagnosis (Ribbens, *et al*, 1994), nuclear power system monitoring (Cheon, *et al*, 1993), chemical processes (Ozyurt & Kandel, 1996), power plant diagnosis (Guglielmi, *et al*, 1995), and diagnosis of bearing faults (Yang, *et al*, 2002). These applications have established that neural networks are a very promising technique.

Among these applications Multi-Layer Perceptrons (MLPs) (Lippmann, 1987, 1989; Rumelhart & McClelland, 1986) have proved the most popular classifier (Zhang, 2000; Sick, 2002), but Radial Basis Function neural Networks (RBFNs) (Broomhead & Lowe, 1988; Moody & Darken, 1988) have also been widely used (Leonard & Kramer, 1990; Musavi, *et al*, 1994; Schwenker, *et al*, 2001).

1.3 Required characteristics of CMFD classifiers

In this study classifiers based on MLPs and RBFNs will be considered. The key question that arises when developing a particular CMFD application is - which of these two classifiers should be used?

Recently several benchmark and comparison studies have been published which have compared the performance of different classifier systems (Michie, *et al*, 1994; Ripley, 1994; Jain & Mao, 1997; Hsu & Lin, 2002). However, three particular problems make the process of classification for CMFD purposes particularly challenging:

1) 'Multiple faults'

   In CMFD applications, simultaneous faults can occur: for example, it is possible that both the failure of a thermostatic valve and a radiator in a cooling system will occur simultaneously, following physical damage. This type of problem was rarely discussed in comparative studies (Rozier, 2001; Sick, 2002) because, in many classifier systems, the problem simply cannot arise: for example, in a speech recognition system, the user might say either 'three' or 'four', but - clearly - cannot say both words simultaneously.

2) 'Unknown faults'

   In CMFD applications, 'unknown faults' can occur: these are conditions that were not anticipated during the classifier design process and which, consequently, were not represented in the datasets used to train the classifier system. Such problems are, again, rarely discussed in comparative studies, because they are uncommon in many classifier systems (Tarassenko, *et al*, 2000).

3) Limited training data

   In CMFD applications, samples representing the normal plant condition are readily available, but samples of fault conditions are often difficult to obtain, especially in safety-critical environments (Bartal, *et al*, 1995; Jack & Nandi, 2001). In addition, obtaining details of fault conditions from manufacturers or suppliers can be difficult.

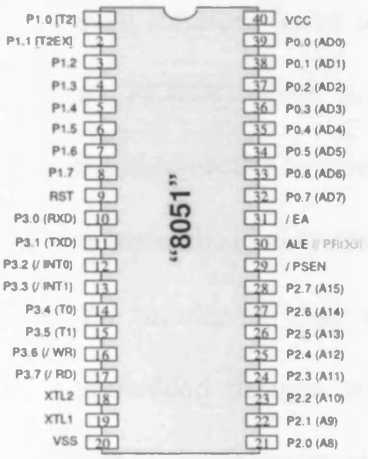
These characteristics of CMFD applications require the classifier designer to consider not only basic classifier performance, but also performance in the context of multiple faults, unknown faults and limited training data. These important features have been largely ignored in previous studies in this area (Dash & Venkatasubramanian, 2000; Sick, 2002).

1.4 Embedded CMFD systems

Where an MLP or RBFN classifier is used in a CMFD application, it is very likely that this classifier will be 'embedded' (for example, see Patton, *et al*, 1989; Isermann & Raab, 1993; Rao, 1996; Marzi, 2002). This fact has a significant impact on the range of system designs that can be economically implemented (Flammini, *et al*, 2001; Dash & Venkatasubramanian, 2000).

Unlike systems implemented on desktop microprocessors (based, for example, on personal computer or workstation architectures), embedded systems (Vahid & Givargis, 2002) have severely limited memory and CPU power (Figure 1-1). For example, a CMFD system based on the widely-used 8051 microcontroller family

will typically have, on each processor node, up to 64 kbytes of external RAM (used mainly to store variables), and up to 64 kbytes of on-chip ROM (used mainly to store the program code). The microcontroller operates at up to 50 million instructions per second (50 MIPS). By comparison, a standard desktop PC will have a multi-gigabyte disk (for program storage), more than 100 Mbytes of RAM (for program execution), and will operate at around 1000 MIPS or more.



Typical features:

- Up to 50 MHz operating frequency: up to 50 million instructions per second (MIPS)
- On-chip data (RAM) memory - 256 bytes.
- On-chip code (ROM) memory - up to 64 kbytes.
- Three sixteen-bit timer / counters.
- Various interrupt sources.
- Cost from ~$0.50 (US), in quantities of 1000+.

Figure 1-1. External interface (left) and features (right) of a small microcontroller. While the features available on different microcontrollers vary considerably, the extensive and popular 8051 family is representative of the type of small microcontroller that might be used to implement an embedded CMFD application of the type discussed in this thesis.

Despite the apparent disadvantages of embedded processors, they are widely used. Indeed more than ten times as many microcontrollers than microprocessors are manufactured and sold in the world (Carelse, 2002). There are three particularly important reasons for this:

- While the desktop PC may have a system board costing $300.00+, the typical cost of the system board for the embedded solution will be of the order of $2.00+. The low cost of microcontroller-based solutions (Flammini, *et al*, 2001) is of central importance in high-volume embedded CMFD applications in, for example, the automotive industry.
- The physical size of a microcontroller-based solution is typically a small fraction of any solution based on a microprocessor (Wilmshurst, 2001); even solutions based on PC architectures designed for embedded use (such as PC/104) have a much larger 'footprint' than the corresponding microcontroller solution. Small physical size is an essential requirement in many embedded systems.
- To provide a true multi-tasking capability and/or redundancy, many embedded designs will employ a network of multiple processors, and CMFD components will be integrated into such systems (Chan, *et al*, 1997). For example, a modern passenger car might contain some forty processors (Leen, *et al*, 1999), controlling brakes, door windows and mirrors, steering, air bags, and so forth. Microcontrollers with Controller Area Network (CAN) and similar communication hardware are readily available (for example, see Lawrenz, 1997), allowing networks of processors to be created with minimal additional hardware complexity, at minimal cost (see Pont, 2001). PC architectures require substantial numbers of additional hardware components, and additional software layers, to implement equivalent designs.

As a result of the consideration of embedded systems, this thesis not only compares the classification performance of MLP and RBFN classifiers, but also considers the costs, in terms of both processor instructions and memory load, of implementing each classifier.

1.5 Pre- and post-processing

The type of CMFD system considered in this thesis usually consists of three stages:

1) the pre-processing of the acquired raw data to extract useful information;
2) the implementation of the classifier to suit the particular application problem;
3) the post-processing of the classifier outputs to provide the state of the plant.

Figure 1-2 illustrates a typical three-stage CMFD system.

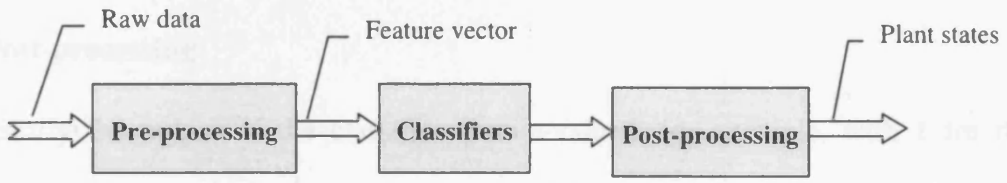


Figure 1-2. A typical three-stage CMFD system.

The pre- and post-processing stages are considered in the sub-sections below.

Pre-processing

Using a CMFD system, the condition of the plant will be inferred from measurements obtained using data-acquisition equipment. The raw data recorded from the plant in this way usually consists of a great number of samples and is likely to contain some level of noise.

In this thesis, the term "pre-processing" will be used to refer to the process of extracting features from this raw data, using appropriate signal processing techniques. After such processing, the plant data will be represented by feature vectors with reduced dimensionality (Manson, *et al*, 2001; Yang, *et al*, 2002).

It should be noted that the dimensionality reduction achieved through pre-processing is of particular importance in embedded CMFD designs (of the type considered in this thesis), not least because it may mean that a simpler/smaller neural classifier can subsequently be employed, with a consequent reduction in CPU and memory requirements.

Post-processing

Ideally the output of the classifier is two-valued, for example, with 1 for the presence of the class and 0 for the absence of the class. However the actual output of a neural classifier rarely appears in this ideal form: that is, elements of the output vector are usually real valued rather than binary valued. The objective of post-processing is to convert the real valued vectors into binary valued vectors in

order to reach the final classification for feature vectors (Theodoridis & Koutroumbas, 1999).

Proper post-processing directly translates into improving classifier system performance (Silipo & Marchesi, 1998; Haykin, 1999). This post-processing stage makes a particularly large contribution to the system performance for some applications such as unknown fault detection (Tarassenko, *et al*, 2000; Hayton, *et al*, 2001) and multi-fault diagnosis (Chung, *et al*, 1994). However the importance of post-processing has been largely overlooked by researchers (Cordella, *et al*, 1995; Lampariello & Sciandrone, 2001).

1.6 Scope of the study

With the characteristics of CMFD applications and the restrictions of embedded systems in mind, this thesis considers the design of CMFD systems as an integrated process of three stages: pre-processing, classifier design and post-processing. The specific aims of this study are as follows:

1) To develop a method for measuring the effectiveness of pre-processing techniques.
2) To evaluate the appropriateness of MLP and RBFN classifiers against criteria that are appropriate for CMFD systems.
3) To identify optimal thresholds for post-processing classifier outputs in RBFN classifiers.

These objectives will be achieved by theoretical derivation and empirical evaluation. The empirical work follows current standard practice (Prechelt, 1996; Jain & Mao, 1997; Zhang, 2000; Hsu & Lin, 2002) in that, each method (or algorithm, classifier, comparison criterion) is evaluated on at least two classification problems.

1.7 Overview of the remainder of this thesis

Following this introductory chapter, Chapter 2 provides an overview of MLP and RBFN classifiers, and their implementation in CMFD systems.

Chapters 3 and 4 are then devoted to the selection of appropriate pre-processing strategies. Chapter 3 first proposes a class-separability measure which is derived from a non-parametric separability analysis: no knowledge of the underlying distribution of the data is required. Chapter 4 then develops a selection procedure for pre-processing techniques using the proposed separability measure. The effectiveness of this procedure is explored on a problem of engine misfire detection.

Chapters 5, 6 and 7 are devoted to the evaluation of MLP and RBFN classifiers, focusing on CMFD applications. Chapter 5 describes how the comparison criteria were selected and provides information about the datasets that were employed in the empirical study. Chapters 6 and 7 describe the empirical studies themselves.

Chapter 8 is devoted to post-processing, focusing on improving RBFN performance where unknown faults may occur. It presents a novel technique that may be used to determine an appropriate threshold for interpreting the outputs of a trained RBFN classifier. Results from two experiments demonstrate that this method can be used to improve the performance of RBFN classifiers in practical applications.

A design methodology is derived in Chapter 9, based on the theoretical and experimental findings from earlier chapters. This methodology is then assessed in a further case study, which involves the design of an embedded CMFD system for a diesel-engine aspiration system.

The final chapter concludes the thesis with an overall discussion, and a presentation of suggestions for future work in this important area.

2
MLP AND RBFN CLASSIFIERS

2.1 Introduction

As discussed in Chapter 1, this thesis explores how two popular neural classifiers, the MLP and RBFN, can be most effectively employed in CMFD systems. This chapter provides essential background information on these two classifiers. It also discusses classifier implementation issues which are particularly important for embedded CMFD applications.

2.2 Multilayer perceptrons

MLPs have been the most widely used neural network (Zhang, 2000). The MLP network consists of an input layer, one or more hidden layers and an output layer, each layer consisting of a number of neurons (nodes). Figure 2-1 illustrates the signal flow in a single neuron which has input signals (x), weights (w), and output signal (y). A bias, w_0, acts exactly as a weight on a connection from a node whose input signal is always 1.

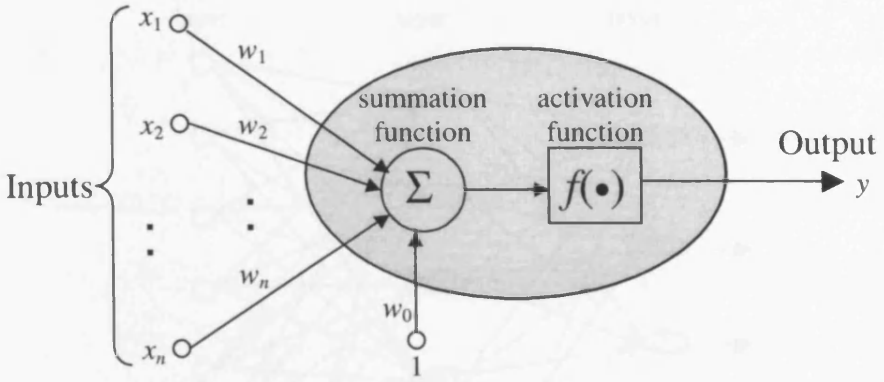


Figure 2-1. Signal flow in a node from an MLP network.

A node carries out two actions. The first is to sum weighted input signals from other nodes, providing the net input for this node:

$$net = \sum_{i=0}^{n} w_i x_i$$

where n is the number of inputs.

The second action is to output a value as a function of its net input:

$$y = f(net)$$

This is commonly carried out using a sigmoid function such as:

$$y = f(net) = \frac{1}{1+e^{-net}}$$

The network structure of an MLP with one hidden layer is depicted in Figure 2-2.

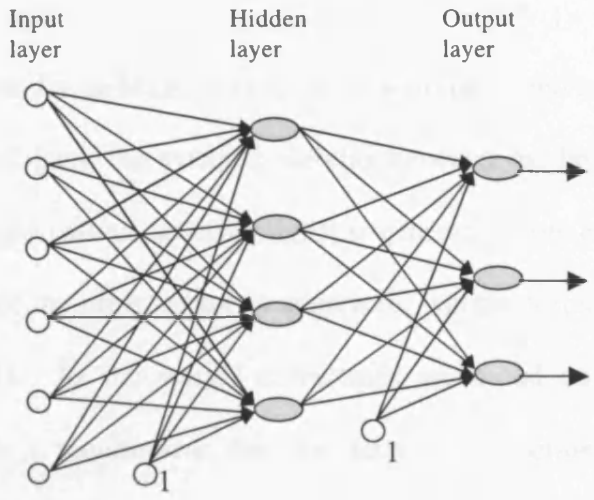


Figure 2-2. An MLP with one hidden layer.

MLPs are trained using the back-propagation (BP) algorithm (Rumelhart & McClelland, 1986). During the learning process, weight corrections are applied according to the delta rule:

$$\Delta w_{ji}(k) = \eta \delta_j(k) x_i(k) \tag{2-1}$$

where, $\Delta w_{ji}(k)$ is the weight correction applied to a synaptic link connecting the neuron j to the neuron i of the preceding layer at the training iteration k.

η is a constant called the learning rate parameter.

$\delta_j(k)$ is a local gradient which depends on whether neuron j is an output neuron or a hidden neuron. If neuron j is an output neuron, $\delta_j(k)$ equals the product of the output error signal (that is, the difference between the target output and the network output) and the derivative of the activation function. If neuron j is a hidden neuron, $\delta_j(k)$ equals the product of the associated derivative of the activation function and the weighted sum of the δs computed for the neurons in the next hidden or output layer that are connected to neuron j.

$x_i(k)$ is the ith input signal of neuron j at the training iteration k.

The training process for an MLP involves a 'forward pass' and a 'backward pass'. During the forward pass, the synaptic weights between the layers are unaltered and, for each training pattern, an error signal is generated from each neuron in the output layer. Once the error vector is generated, weight corrections are applied according to (2-1). As the weight corrections are based on a local gradient computation, it is a requirement that the activation functions used for MLP networks are differentiable.

Note that training of the network can be undertaken in pattern mode (weight changes are calculated and applied after each training pattern has been presented) or in batch mode (weight changes are calculated and applied after presentation of all the training patterns: the presentation of the whole set of training patterns is known as an *epoch*).

The learning rate η in (2-1) determines the size of weight changes in any given iteration. If the learning rate is set too low, the training time will increase to an unacceptable duration. On the other hand, if the learning rate is set too high, large weight changes will occur which make the network unstable, oscillate and fail to converge.

Direct use of this gradient descent algorithm is particularly inefficient in training MLPs, and various modifications have been proposed (Haykin, 1999). A method

to improve training efficiency and yet avoid the occurrence of instability is to modify the delta rule by adding a momentum term (Bishop, 1995):

$$\Delta w_{ji}(k) = \eta \delta_j(k) x_i(k) + \alpha \Delta w_{ji}(k-1) \qquad (2\text{-}2)$$

where α is usually a positive number called the momentum constant.

The momentum term may also have the benefit of preventing the learning process from terminating in a shallow local minimum on the error surface.

Equation (2-2) is called the generalised delta rule containing two (arbitrary) parameters, η and α, whose values must be adjusted to give the best performance. Furthermore, the optimal values for these parameters will often vary during the training process. One effective strategy for varying the learning rate is to compare the error ratio of successive iteration steps, $\frac{e(k)}{e(k-1)}$ (Haykin, 1999). If the error at the kth iteration step, $e(k)$, is smaller than the previous error $e(k\text{-}1)$, then the learning rate is increased. If, on the other hand, $e(k)$ is greater than $e(k\text{-}1)$ by more than a predefined ratio, the learning rate is decreased.

2.3 Radial basis function networks

The radial basis function is a functional approximation technique (Powell, 1987). Given m different points $\{\mathbf{x}_i : i = 1,2,\cdots,m\}$ in R^n, and m real numbers $\{y_i : i = 1,2,\cdots,m\}$, one has to calculate a function f: $R^n \rightarrow R$ that satisfies the interpolation condition:

$$f(\mathbf{x}_i) = y_i \qquad i = 1,2,\cdots,m$$

Powell (1987) chose f from a linear space that depends on the positions of the data points. The positions of data points is described using radial basis functions of the form:

$$\phi(\|\mathbf{x} - \mathbf{x}_i\|), \qquad \mathbf{x} \in R^n \quad i = 1,2,\cdots,m \tag{2-3}$$

where $\|\bullet\|$ is the norm of R^n.

Table 2-1 lists some examples of radial basis function nonlinearities (Sanchez, 1996).

Function name	Function expression: $\phi(r, \sigma)$ (σ=constant)
Linear	r
Cubic	r^3
Thin plate spline	$r^2.\log r$
Gaussian	$\exp(-[r^2/\sigma^2])$
Multiquadric	$(r^2+\sigma^2)^{\pm 1/2}$

Table 2-1. Examples of radial basis function nonlinearities.

The approximation functions have the form:

$$f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi(\|\mathbf{x} - \mathbf{x}_i\|), \qquad \mathbf{x} \in R^n \tag{2-4}$$

Broomhead & Lowe (1988) and Moody & Darken (1988) first independently converted this functional approximation technique into a neural network paradigm. The resulting RBFNs consist of one input, one hidden and one output layer (Figure 2-3).

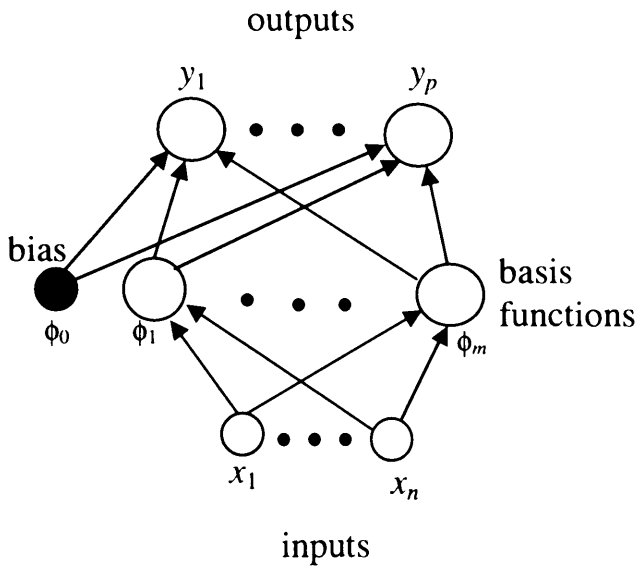


Figure 2-3. The architecture of RBFNs.

In Figure 2-3 the hidden layer contains nodes realising basis functions while the output layer nodes form a linear combination of the hidden layer outputs. Thus an output of the RBFN, given input vector **x**, may be described as follows:

$$\begin{aligned} y_k(\mathbf{x}) &= \sum_{i=0}^{m} w_{ik}\phi_i(\|\mathbf{x}-\mathbf{c}_i\|) \\ &= \sum_{i=1}^{m} w_{ik}\phi_i(\|\mathbf{x}-\mathbf{c}_i\|) + b_k \end{aligned} \qquad k = 1,2,\cdots p \qquad (2\text{-}5)$$

where $\mathbf{c}_i$ is the centre of ϕ_i (ϕ_i could be one of the expressions in Table 2-1), w_{ik} is the weight connecting the ith hidden node and the kth output node, p is the

number of output neurons, the radial basis function ϕ_0 is set to be a constant equal to 1, and bias $b_k = w_{0k}$.

For a set of input vectors $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_N\}$, Equation (2-5) can be rewritten as:

$$\mathbf{y}_k = \Phi \mathbf{w}_k \tag{2-6}$$

where:

$$\mathbf{y}_k = [y_k(\mathbf{x}_1), y_k(\mathbf{x}_2) \cdots, y_k(\mathbf{x}_N)]^T$$

$$\mathbf{w}_k = [b_k, w_{1k}, w_{2k}, \cdots, w_{mk}]^T$$

$$\Phi = \begin{bmatrix} 1 & \phi_1(\|\mathbf{x}_1 - \mathbf{c}_1\|) & \phi_2(\|\mathbf{x}_1 - \mathbf{c}_2\|) & \cdots & \phi_m(\|\mathbf{x}_1 - \mathbf{c}_m\|) \\ 1 & \phi_1(\|\mathbf{x}_2 - \mathbf{c}_1\|) & \phi_2(\|\mathbf{x}_2 - \mathbf{c}_2\|) & \cdots & \phi_m(\|\mathbf{x}_2 - \mathbf{c}_m\|) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \phi_1(\|\mathbf{x}_N - \mathbf{c}_1\|) & \phi_2(\|\mathbf{x}_N - \mathbf{c}_2\|) & \cdots & \phi_m(\|\mathbf{x}_N - \mathbf{c}_m\|) \end{bmatrix}$$

$$= [\mathbf{1} \quad \boldsymbol{\phi}_1 \quad \boldsymbol{\phi}_2 \quad \cdots \quad \boldsymbol{\phi}_m]$$

where T denotes transpose.

For the Gaussian function with centre $\mathbf{c}_i$ and width $\sigma_i (i = 1,2,\cdots,m)$, ϕ_i has the form:

$$\phi_i(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}_i\|^2}{2\sigma_i^2}\right) \tag{2-7}$$

The basis function (nonlinearity) in the hidden layer produces a significant non-zero response to input stimuli only when the input falls within a small localised

region of the input space, and the RBFNs form a bounded decision region (Lampariello & Sciandrone, 2001).

Since the hidden layer and output layer of an RBFN perform different tasks, most of the learning algorithms consist of two stages. The first stage selects the appropriate centres of the radial basis functions, and the second estimates the weights between the hidden layer and the output layer using linear optimisation algorithms.

The centre can be selected using the following algorithms:

1) Fix the number of radial basis functions, m ($m \leq N$, where N is the number of samples in training dataset), in the hidden layer, and choose locations of the m centres randomly from the training dataset. In order to achieve effective RBFNs, this approach requires that the training data well represents the data distribution of the problem at hand (Moody & Darken, 1988).
2) Adapt the centres of the radial basis functions in a self-organised fashion such as k-nearest neighbour rule or other clustering methods (Lee & Kil, 1991).
3) Gradually optimise the centres of the radial basis functions by supervised learning using the gradient descent procedure (Tarassenko & Roberts, 1994; Orr, 1999).

When we design an RBFN, it is desirable to obtain a parsimonious model. One effective way to achieve this objective is to use the *orthogonal least squares* (OLS) learning algorithm (Chen, *et al*, 1991). The OLS learning algorithm will be used in the following chapters and therefore deserves a more detailed discussion.

To apply the OLS method to design RBFN, Equation (2-6) is viewed as a special case of the linear regression model as follows (Chen, *et al*, 1991):

$$\mathbf{y} = \Phi\mathbf{w} + E \tag{2-8}$$

where $\mathbf{y} \in R^N, \Phi \in R^{N\times m}$ and $\mathbf{w} \in R^m$ (for simplicity, here m is equivalent to m+1 in (2-6)) are the same as defined in (2-6), E is the error signal which is uncorrelated with the regressor vectors ϕ_i:

$$E = [e_1, e_2, \cdots, e_N]^T \in R^N$$

The regressor vectors ϕ_i form a set of basis vectors, and the least squares solution $\hat{\mathbf{w}}$ satisfies the condition that $\Phi\hat{\mathbf{w}}$ be the projection of $\mathbf{y}$ onto the space spanned by these basis vectors.

The OLS method involves the transformation of the regressor vectors ϕ_i into a corresponding set of orthogonal vectors $\mathbf{u}_i$ (i=1, 2, ..., m), that is, the regression matrix Φ can be decomposed into:

$$\Phi = \mathbf{UA} \tag{2-9}$$

where $\mathbf{A} \in R^{m \times m}$ is an upper triangular matrix with 1's on the diagonal and 0's below the diagonal, and $\mathbf{U} \in R^{N \times m}$ is a matrix with orthogonal columns $\mathbf{u}_i$ such that:

$$\mathbf{u}_i \mathbf{u}_j = 0, \quad \text{for} \quad i \neq j \tag{2-10}$$

The orthogonal decomposition of Φ into $\mathbf{U}$ and $\mathbf{A}$ can be obtained using the Householder transformation method, classical Gram-Schmidt algorithm or the modified Gram-Schmidt algorithm (Watkins, 1991). In the classical Gram-Schmidt algorithm, the transformation is performed as follows:

$$\left.\begin{aligned} \mathbf{u}_1 &= \boldsymbol{\phi}_1 \\ a_{ik} &= \frac{\mathbf{u}_i^T \boldsymbol{\phi}_k}{\mathbf{u}_i^T \mathbf{u}_i}, \qquad 1 \le i < k \\ \mathbf{u}_k &= \boldsymbol{\phi}_k - \sum_{i=1}^{k-1} a_{ik} \mathbf{u}_i \end{aligned}\right\} \quad k = 2,3,\cdots,m \tag{2-11}$$

where the transformation is computed column by column.

The space spanned by the set of orthogonal basis vectors $\mathbf{u}_i$ is the same space spanned by the set of ϕ_i , Equation (2-8) can therefore be rewritten as:

$$\mathbf{y} = \mathbf{U}\mathbf{g} + E \tag{2-12}$$

where $\mathbf{g} \in R^m$. The OLS solution $\hat{\mathbf{g}}$ of $\mathbf{g}$ is then given by:

$$\hat{\mathbf{g}} = (\mathbf{U}^T \mathbf{U})^{-1} \mathbf{U}^T \mathbf{y} \tag{2-13}$$

or

$$\hat{g}_i = \mathbf{u}_i^T \mathbf{y} / (\mathbf{u}_i^T \mathbf{u}_i) \qquad \text{for } i=1, \ldots, m$$

The quantities $\hat{\mathbf{w}}$ and $\hat{\mathbf{g}}$ satisfy:

$$\mathbf{A}\hat{\mathbf{w}} = \hat{\mathbf{g}} \tag{2-14}$$

where $\hat{\mathbf{w}}$ is the estimate of $\mathbf{w}$ in (2-8).

Using (2-10), the sum of squares of $\mathbf{y}$ can be obtained from (2-12) and is given by:

$$\mathbf{y}^T \mathbf{y} = \sum_{i=1}^{m} g_i^2 \mathbf{u}_i^T \mathbf{u}_i + E^T E \tag{2-15}$$

Equation (2-15) shows how an individual regressor contributes to the sum of squares of $\mathbf{y}$. The error reduction ratio due to $\mathbf{u}_i$ can be defined as:

$$err_i = \frac{g_i^2 \mathbf{u}_i^T \mathbf{u}_i}{\mathbf{y}^T \mathbf{y}} \tag{2-16}$$

Using (2-16), the significant regressors can be selected.

In the design of RBFNs, the OLS learning procedure selects the radial basis function centres as a subset of the N training data vectors. At the kth step of the selection procedure, a candidate centre $\mathbf{c}_k$ is selected as forming the kth regressor centred at $\mathbf{c}_k$ if the regressor produces the largest value of err_k from among the rest of N-k+1 candidate centres. The selection process is terminated when the inequality (2-17) becomes true:

$$1-\sum_{j=1}^{k} err_j < \rho \qquad (2\text{-}17)$$

where $0 < \rho < 1$ is a chosen tolerance. This gives rise to a subset network containing m significant hidden nodes.

The OLS learning procedure generally produces an RBFN whose hidden layer is smaller than an RBFN with randomly selected centres. It therefore provides a useful means for the construction of a parsimonious RBFN (Chen, *et al*, 1991).

2.4 Classifier parameters and implementation

This section discusses the implementation of the neural network classifiers which are applied throughout the remainder of this thesis.

2.4.1 Parameters for classifier structure

The structure of neural network classifiers is designed by the number of input nodes, hidden layers, hidden nodes and output nodes.

Input nodes

The number of input nodes in a neural classifier is determined by the dimensions of the feature space. For an n-dimensional feature space, n input nodes are needed. Thus if, for example, a vibration signature to be classified is 1000

samples long (in the time domain), a neural network with 1000 input nodes is required[1].

Number of hidden layers

The determination of the number of hidden layers is only relevant to MLP design as the RBFN was originally developed with only one hidden layer. MLPs with one hidden layer are used throughout this study as in most published applications. This is because MLPs with just one hidden layer (with enough hidden nodes) can form decision regions with arbitrary shapes, being universal approximators for arbitrary finite-input environment measures (Huang & Lippmann, 1988; Huang, *et al*, 2000).

Number of hidden nodes

The number of hidden neurons in an MLP has a great impact on the network's classification performance. Huang & Huang (1991) have argued that, for the finite subset of feature space R^n of a multi-input-single-output system, the optimal number of hidden neurons m to realise an arbitrary function is $m = N - 1$, where N is the number of input patterns. In practice, fewer than m hidden neurons are generally required since real datasets often consist of redundant patterns. Indeed there are no general rules to decide the optimal number of hidden neurons (Setiono, 2001; Leung, *et al*, 2003).

---

[1] In most CMFD applications, this approach is impractical, and raw sensor data must be pre-processed before application to the classifier. Selection of an appropriate pre-processing technique will be discussed in Chapter 4.

In this thesis, in the absence of a better solution and in line with previously published works, a simple 'trial-and-error' approach was employed to determine the most appropriate number of hidden nodes in the MLP classifiers. Specifically, networks were trained with different numbers of hidden neurons (using a training dataset) and then the error for each was calculated (using a different dataset). The network with the minimum testing error dictated the 'optimal' number of hidden nodes.

For RBFNs, the number of hidden neurons may be determined either by a hybrid learning algorithm (Moody & Darken, 1989) or by an OLS learning algorithm (described in Section 2.3). In both cases, a heuristic-based selection of the widths can be incorporated in order to achieve a certain amount of response overlap between each node and its neighbours. This ensures that, when combined, the nodes form a smooth and continuous interpolation over those regions of the input space. In this study, OLS learning algorithm was used. This algorithm expands the hidden layer by adding a new neuron at each of the learning iterations. The new hidden neuron is added at the location within learning data so that it reduces the leaning error signal of the classifier (Chen, *et al*, 1991).

Number of output nodes

The number of output nodes is set to be equal to the number of known classes, with a single output node for each known class. Details of the representation scheme are given in the next section.

2.4.2 Interpretation of classifier outputs

The output classes can be represented by the network in several ways. In the majority of the experiments in this thesis, a single output neuron is used to represent each class. For example, an output of {'1', '0', '0'} from a three-output network would indicate a classification of 'Class 1', an output of {'0', '1', '0'} would indicate a classification of 'Class 2', and {'0', '0', '1'} would indicate 'Class 3'.

In practice, such idealised outputs are rarely observed, and the outputs of the classifier must be processed to give the final decision. Figure 2-4 depicts the process of assigning an input pattern $\mathbf{x}_i$ from neural classifier outputs, where a is {0,1}.

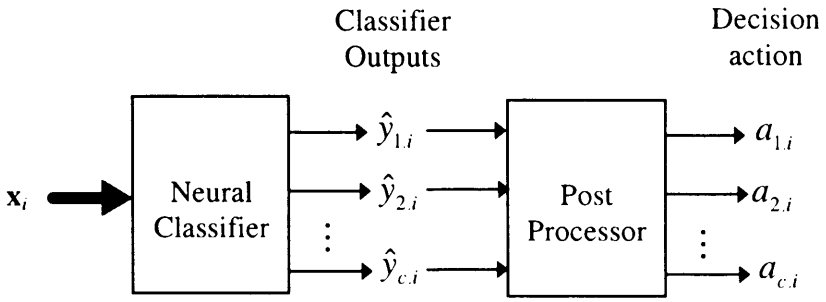


Figure 2-4. Conversion of classifier outputs to decision action.

It is usually assumed that the largest output represents the class output. Thus the post-processor is defined by:

$$a_{k,i} = \begin{cases} 1 & \text{if} \quad \hat{y}_{k,i} = \max\limits_{k=1,\cdots,c} \left(\hat{y}_{k,i}\right) \\ 0 & \text{otherwise} \end{cases} \qquad \text{for} \quad k = 1,\cdots,c$$

The value of $a_{k,i}$ gives a decision action where the input pattern $\mathbf{x}_i$ is assigned to the kth class if $a_{k,i}=1$.

This simple post-processing strategy is clearly not appropriate for CMFD applications where unknown-fault and multiple-fault may occur. Instead, a threshold, τ, may be applied to classifier outputs. It is then understood that the output represents the class presence if its value exceeds the threshold. Thus the post-processor for such CMFD applications is defined by:

$$a_{k,i} = \begin{cases} 1 & \text{if} \quad \hat{y}_{k,i} > \tau \\ 0 & \text{if} \quad \hat{y}_{k,i} \leq \tau \end{cases} \qquad \text{for} \quad k = 1, \cdots, c$$

This allows the neural classifier to represent the occurrence of multiple and unknown faults. For example, an output of {'0', '1', '1'} would indicate an output in 'Class 2' and 'Class 3' simultaneously, and an output vector which does not match any output pattern in training data would indicate the occurrence of unknown faults.

As we have discussed above, it is possible that 'unknown faults' will be encountered, where classifier inputs are not represented in the training dataset. When a classifier is used in such applications, there are several possible ways of representing these faults, including the following:

1) To use an extra output neuron to represent unknown faults, so that the number of output neurons is equal to the number of known classes plus one;

2) To set the number of output neurons equal to the number of known classes.

A brief analysis suggests that the first approach is impractical. Recall that the response of an output neuron is determined as follows:

$$\text{for MLP:} \quad y_k = f(net_k) = \frac{1}{1+e^{-net_k}}, \quad net_k = \mathbf{w}_k^{\mathrm{T}}\mathbf{y}_h + b_k \tag{2-18}$$

$$\text{for RBFN:} \quad y_k = \mathbf{w}_k^{\mathrm{T}}\Phi + b_k \tag{2-19}$$

where $\mathbf{w}_k$ is a vector of weights from the hidden neurons to the output neuron k, $\mathbf{y}_h$ is a vector of outputs of the hidden neurons (subscript h for hidden layer), b_k is a bias. When the classifiers are trained using data of known classes, the value of the 'unknown fault' output neuron is required to be assigned to a constant (denoted *const*) to indicate that an unknown fault is not present. Thus, during training, $\mathbf{w}_k$ will simply be assigned to zero and b_k to $f^{-1}(const)$ for the MLP, and b_k to *const* for the RBFN. The trained classifiers will therefore produce *const* on the 'unknown fault' output neuron no matter what value the input samples take.

Since an extra output neuron cannot provide a useful representation of unknown faults, it is more appropriate to set the number of output neurons equal to the number of known classes. In this situation, the classifier outputs need to be combined and interpreted in a slightly different way to that described above:

- If one output exceeds a value of τ (τ is a pre-set threshold), then we assume that this output identifies the fault.

- If the classifier output vector is not the same as the desired output vector, then the input vector is misclassified, and the corresponding sample is included in the "error" category.
- If no output exceeds τ, the input is assumed to be an 'unknown fault'.

In such a CMFD application, one of the key issues is then to determine the value of the threshold. The value of τ can be determined by analysis of the bias values in the output layer[2].

2.4.3 Estimation of error rates

In a classification problem, the designer has a set of data samples $\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_N$. Each sample consists of two parts $\mathbf{z}_i = (\mathbf{x}_i, \mathbf{y}_i)$, where $\mathbf{x}_i$ is a vector of input variables and $\mathbf{y}_i$ is the corresponding vector indicating the presence or absence of classes.

The original dataset may then be re-organised into j subsets. Each subset has N_j samples, where usually $N = \sum_j N_j$, for example for j=2, there are N_1 samples in the training dataset and N_2 samples in the test set.

On the basis of the training dataset, a classifier $\hat{\mathbf{y}} = \mathcal{N}(\mathbf{x})$ is constructed ($\mathcal{N}$ denotes the classifier and $\hat{\mathbf{y}}$ is the estimate of $\mathbf{y}$) and its outputs are converted to a class

---

[2] This process is discussed further in Chapter 8.

decision, **a**, using the post-processing scheme discussed in the previous section. The objective is then to estimate the classifier performance using an (unseen) dataset independent of the training dataset. The classification error rate on the dataset of N_j samples is defined as:

$$\varepsilon = \frac{1}{N_j} \sum_{i=1}^{N_j} L(\mathbf{y}_i, \mathbf{a}_i)$$

where L is a loss function defined by:

$$L(\mathbf{y}_i, \mathbf{a}_i) = \begin{cases} 0 & \text{if} \quad \mathbf{y}_i = \mathbf{a}_i \\ 1 & \text{otherwise} \end{cases}$$

This estimation can be achieved by one of the following error estimation methods: the holdout method (Bishop, 1995), the cross validation method (Theodoridis & Koutroumbas, 1999) or the bootstrap method (Efron, 1979; Hamamoto, *et al*, 1997), based on the organisation of sub-sets of data and the estimation procedure.

In this thesis, the holdout method is used for designing classifiers. This method involves dividing each dataset into two parts, usually with equal size, one for training and the other for testing. This approach is taken because the goal of this study is to compare the performance of the classifiers, rather than to provide optimal performance results for each problem. Provided the comparison conditions are the same for all classifiers, the results will indicate the relative performance of individual classifiers. Therefore, in classifier comparisons, the error estimation method is not important so long as the error estimation method is the same for all classifiers (Michie, *et al*, 1994; Blayo, *et al*, 1995).

2.4.4 Activation functions and learning algorithms

Each of the classifiers used in this study was implemented in its most commonly used form. Specifically, the MLPs were implemented with one hidden layer using logistic sigmoid functions for both the hidden layer and the output layer (Haykin, 1999). Similarly, the RBFNs were implemented using Gaussian functions for the hidden layer and linear functions for the output layer (Looney, 1997).

Both MLPs and RBFNs were implemented using the Neural Network Toolbox version 2.0.4 in Matlab[3].

The MLPs were trained using Matlab's "trainbpx" which is widely used (Duin, 1996), setting the maximum number of 'epochs' to 4000 and using default values for the following training parameters: learning rate increase (1.05); learning rate decrease (0.7); momentum constant (0.9) and maximum error ratio (1.04).

The RBFNs were designed using the "solverb" function which implements the orthogonal least square learning algorithm proposed by Chen *et al* (1991). Please note that the commonly used term 'width' for the radial basis function is equivalent to the term 'spread constant' in the Neural Network Toolbox.

---

[3] All implementations were in Matlab, with the exception of the memory experiments discussed in Chapter 6.

2.4.5 Implementing the embedded classifiers

This section briefly considers how neural classifiers might be implemented in an embedded system.

In order to implement such applications, a network of at least two low-cost microcontrollers (for example, from the 8051 family) can be made use of, which are integrated by means of a shared-clock scheduler architecture (see Pont, 2001, for further details of the scheduler architecture).

The resulting system architecture is illustrated schematically in Figure 2-5.

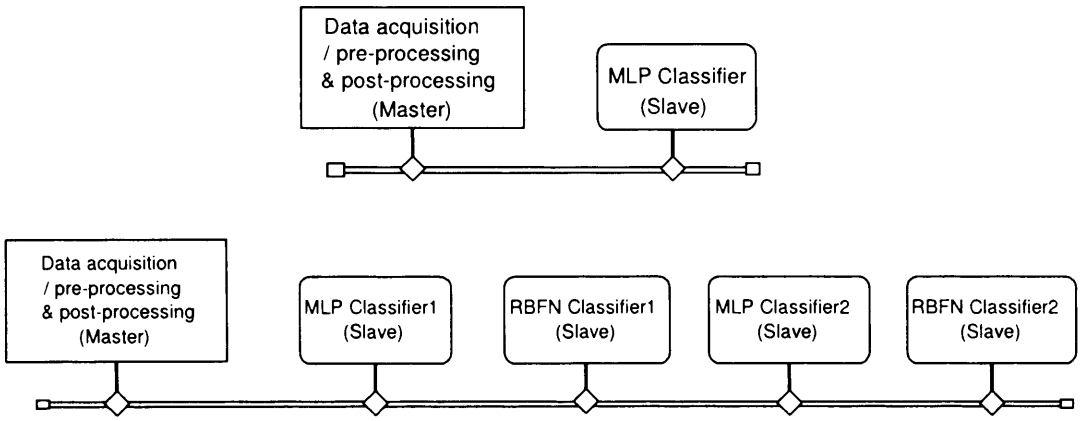


Figure 2-5. Two possible embedded architectures that might be used to implement the classifier systems discussed in this study. In each case, more than one microcontroller is used and the operations are synchronised by means of a time-triggered software architecture, realised using a shared-clock scheduler.
(Top) A simple architecture with two microcontrollers, used to implement a single classifier with data acquisition and pre- and post-processing.
(Bottom) A more complex architecture used to implement a fusion classifier with both MLP and RBFN components.

2.5 Conclusions

The purpose of this chapter was to introduce the two classifiers used throughout this thesis. The back-propagation algorithm for training MLP was briefly introduced. An essential part of the design of a RBFN classifier is how to select radial basis function centres from the training set. This chapter described the orthogonal least squares learning algorithm which is an efficient and commonly used training algorithm for RBFN classifiers.

In the use of MLP and RBFN for CMFD applications, a number of network parameters need to be determined. In order to use the classifier for cases of unknown and multiple faults, an interpretation strategy for classifier outputs was discussed. This chapter also briefly considers how the studied classifiers could be implemented in an embedded system.

Following the review of MLP and RBFN classifiers presented here, Chapter 3 will begin to address the first stage of designing embedded CMFD systems.

3
A SEPARABILITY MEASURE FOR DATA WITH AN UNRESTRICTED DISTRIBUTION

3.1 Introduction

As discussed in Chapter 1 the 'raw' signals obtained from sensors are rarely applied directly to the classifier. Instead these raw signals are pre-processed prior to classification (Theodoridis & Koutroumbas, 1999; Liu, *et al*, 2002).

Numerous forms of pre-processing techniques may be employed (Petrilli, *et al*, 1995; Tsoi & Back, 1995; Yang, *et al*, 2002), but the most common involve some form of filter (for example, low-pass, notch or moving-average), and/or some form of transform (for example, decimation, Fourier transform or wavelet transform). Despite the variation in techniques, the three aims of pre-processing are generally similar (Staszewski & Worden, 1997; Theodoridis & Koutroumbas, 1999; Somol & Pudil, 2002; Sick, 2002):

1) To enhance the difference between examples from different classes;
2) To minimise the difference between examples from the same class;
3) To reduce the size of the dataset, allowing the use of a smaller neural network (with faster responses and reduced memory requirements).

Since the designers of CMFD systems have a range of different types of pre-processing techniques at their disposal, there is a need to have a measure to indicate the effectiveness of individual techniques in order to identify the most appropriate pre-processing approach.

It seems reasonable to assume that a dataset which contains highly separated classes would be easier to classify than one containing overlapping classes (Blayo, *et al*, 1995; Theodoridis & Koutroumbas, 1999). Thus, if we are able to obtain a suitable measure of class separability, we might reasonably expect that this would form the basis of an effective means of comparing pre-processing techniques.

Although there are some statistical measures for measuring separability between classes (Fukunaga, 1990), they are rarely used in CMFD applications. This is because existing separability measures assume that the probability distributions of the dataset are known (Theodoridis & Koutroumbas, 1999), but in real CMFD applications the distributions are unlikely to be known *a priori* (Heinke & Hamker, 1998; Marzi, 2002). To overcome the limits of existing measures, this chapter proposes a new method for measuring the separability for datasets with unknown probability distributions. This separability measure is then applied in a procedure for identifying effective pre-processing techniques in Chapter 4.

3.2 Classical separability analysis

Measures for describing datasets may be divided into three main categories:

1) Simple measures such as: the number of attributes, the number of classes, and the dataset size;
2) Statistical measures such as: those based on class distributions and dataset density (Blayo, *et al*, 1995; Theodoridis & Koutroumbas, 1999);
3) Information theoretic measures such as: entropy and information scoring (Zheng, 1993).

Recent research has revealed that neural networks have a strong link to statistics (Cheng & Titterington, 1994; Ripley, 1994; Kay & Titterington, 1999). If a statistical measure is able to sufficiently describe the distributional features of a dataset, it may be used to predict the classification difficulty of the dataset for neural networks (Blayo, *et al*, 1995; Heinke & Hamker, 1998).

In this study, therefore, the emphasis is placed on describing a dataset in terms of statistics. In particular, the focus is on an analysis of class separability for a given dataset. Class separability provides a measure of the extent to which samples of different classes overlap. Classical separability measures are introduced first in order to examine their limitations when applied to CMFD applications.

3.2.1 Classical separability measures

Some useful criteria for separability analysis are the *Fisher criterion*, *divergence* and the *Bhattaharyya distance* (Heinke & Hamker, 1998). Before considering

these measures, the concept of *inertia* is introduced. Inertia is a classical measure of the variance of data[4].

Inertia

Consider a dataset of N samples with c classes. For the class ω_i (for i=1,...c), the number of samples is N_i , the *a priori* probability is P_i and the mean is $\mathbf{m}_i$. We have:

class ω_i *inertia*:

$$I_{\omega_i} = \frac{1}{N_i}\sum_{j=1}^{N_i}\left\|\mathbf{x}_j - \mathbf{m}_i\right\|^2; \quad \text{for} \quad class(\mathbf{x}_j) = \omega_i \text{ , for all } \mathbf{x} \tag{3-1}$$

within-class inertia:

$$I_w = \sum_{i=1}^{c} P_i I_{\omega_i} = \frac{1}{N}\sum_{i=1}^{c} N_i I_{\omega_i} \tag{3-2}$$

between-class inertia:

$$I_b = \frac{1}{N}\sum_{i=1}^{c} N_i \left\|\mathbf{m}_i\right\|^2 \tag{3-3}$$

where $\|\cdot\|$ is the Euclidean norm: $\|\mathbf{x}\| = (\mathbf{x}^T \cdot \mathbf{x})^{1/2}$.

---

[4] Details of the measures in this sub-section are available from Fukunaga (1990).

Fisher criterion

The Fisher criterion measures the between-class inertia normalised by the within-class inertia and is defined by:

$$FC = \frac{I_b}{I_w} \tag{3-4}$$

The separability gets better as the Fisher criterion (FC) gets larger.

Dispersion

The mean dispersion of class ω_i in class ω_j is defined by:

$$D(i,j) = \frac{\left\| \mathbf{m}_i - \mathbf{m}_j \right\|}{\sqrt{I_{\omega_i}}} \tag{3-5}$$

If the dispersion measure between two classes is large, these classes are well separated.

Bhattaharyya distance

The Bhattaharyya distance between two classes is defined by:

$$B = \frac{1}{8}(\mathbf{m}_2 - \mathbf{m}_1)^T \left(\frac{\Sigma_1 + \Sigma_2}{2} \right)^{-1} (\mathbf{m}_2 - \mathbf{m}_1) + \frac{1}{2} \ln \frac{\left| \frac{\Sigma_1 + \Sigma_2}{2} \right|}{\sqrt{|\Sigma_1| \cdot |\Sigma_2|}} \tag{3-6}$$

where $|\mathbf{x}|$ is the determinant of matrix $\mathbf{x}$ and Σ_i is the covariance matrix of class ω_i.

As seen in (3-6), the Bhattaharyya distance consists of two terms. The first or second term disappears when $\mathbf{m}_1 = \mathbf{m}_2$ or $\Sigma_1 = \Sigma_2$, respectively. Therefore, the

first term measures the class separability due to the mean difference, while the second term measures the class separability due to the covariance difference.

Divergence

If the distributions of two classes are normal, the divergence is defined by:

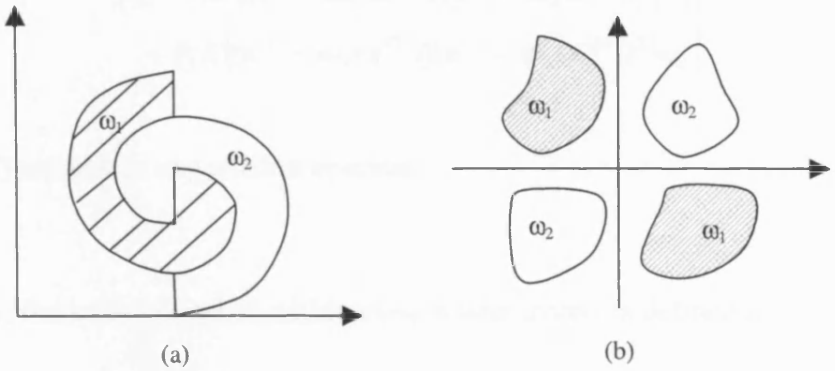
$$D = \frac{1}{2}\mathrm{tr}\left\{\left(\Sigma_1^{-1} + \Sigma_2^{-1}\right)\left(\mathbf{m}_2 - \mathbf{m}_1\right)\left(\mathbf{m}_2 - \mathbf{m}_1\right)^T\right\} + \frac{1}{2}\mathrm{tr}\left(\Sigma_1^{-1}\Sigma_2 + \Sigma_2^{-1}\Sigma_1 - 2\mathbf{I}\right) \quad (3\text{-}7)$$

The divergence also consists of two terms similar to that of Bhattaharyya distance. However both terms of the divergence are expressed by the trace of a matrix.

3.2.2 Limitations of classical separability measures

The aforementioned parametric measures are all based on the assumption that the class distributions are known *a priori* to be normal (Fukunaga, 1990; Theodoridis & Koutroumbas, 1999). When developing a practical CMFD application of the type considered in this thesis, the class distributions are unlikely to be known until sufficient knowledge about the faults is available (Tarassenko, *et al*, 2000; Lada, *et al*, 2002; Skoundrianos & Tzafestas, 2002).

If class distributions are significantly non-normal and multimodal (see, for example, Figure 3-1), the use of parametric measures cannot be expected to accurately indicate the class separability. For this reason it is highly desirable to have a separability measure that does not require any assumptions about class distributions.



(a) (b)

Figure 3-1. Examples of classification problems with non-normal and multimodal distributions.
(a)The classification problem involves nonconvex decision regions (Haykin, 1999).
(b) A typical nonlinear classification problem represents a range of mechanical system fault diagnosis. This example can be found in many publications.

3.3 The proposed non-parametric separability measure

To overcome the limits of classical measures, an alternative non-parametric separability measure is more appropriate in circumstances where the data distribution is unknown. Based on the idea of non-parametric discriminant analysis (Fukunaga, 1983; 1990), the proposed separability measure is described below.

For a two-class problem, let class 1 be ω_1 and class 2 be ω_2, the *a priori* probability of class ω_i is P_i, then the non-parametric between-class scatter matrix is defined as:

$$\begin{aligned}\mathbf{S}_b = {} & P_1 E\left\{(\mathbf{x}^{(1)} - \mathbf{m}_2(\mathbf{x}^{(1)}))(\mathbf{x}^{(1)} - \mathbf{m}_2(\mathbf{x}^{(1)}))^T \middle| \omega_1\right\} \\ & + P_2 E\left\{(\mathbf{x}^{(2)} - \mathbf{m}_1(\mathbf{x}^{(2)}))(\mathbf{x}^{(2)} - \mathbf{m}_1(\mathbf{x}^{(2)}))^T \middle| \omega_2\right\}\end{aligned} \tag{3-8}$$

where $E()$ represents expectation operator.

Similarly, the non-parametric within-class scatter matrix is defined as:

$$\begin{aligned}\mathbf{S}_w = {} & P_1 E\left\{(\mathbf{x}^{(1)} - \mathbf{m}_1(\mathbf{x}^{(1)}))(\mathbf{x}^{(1)} - \mathbf{m}_1(\mathbf{x}^{(1)}))^T \middle| \omega_1\right\} \\ & + P_2 E\left\{(\mathbf{x}^{(2)} - \mathbf{m}_2(\mathbf{x}^{(2)}))(\mathbf{x}^{(2)} - \mathbf{m}_2(\mathbf{x}^{(2)}))^T \middle| \omega_2\right\}\end{aligned} \tag{3-9}$$

Here, samples $\mathbf{x}^{(i)} \in \omega_i$ and $\mathbf{m}_i(\mathbf{x}^{(l)})$ is the ω_i-local mean for a given sample $\mathbf{x}^{(l)}$, computed from the k-nearest neighbours in ω_i to $\mathbf{x}^{(l)}$:

$$\mathbf{m}_i(\mathbf{x}^{(l)}) = \frac{1}{k}\sum_{j=1}^{k} \mathbf{x}_{jNN}^{(i)} \tag{3-10}$$

Note that, when $\mathbf{S}_w$ is computed, it is necessary to exclude the sample $\mathbf{x}^{(l)}$ from our k-NN determination, as $\mathbf{x}^{(l)}$ should not be considered a nearest neighbour to itself. Moreover, in the calculation of $\mathbf{S}_w$, all samples which are situated at the same position as $\mathbf{x}^{(l)}$ must be excluded as the nearest neighbours of $\mathbf{x}^{(l)}$ in order to avoid distorting the local information.

Figure 3-2 illustrates the calculation of local mean for between-class scatter with k=2. The derivation of local means in class ω_2 for two samples $\mathbf{x}_1^{(1)}$ and $\mathbf{x}_2^{(1)}$ in class ω_1 is also shown in the figure. For $\mathbf{x}_1^{(1)}$ the two nearest neighbours in ω_2 are $\mathbf{x}_1^{(2)}$ and $\mathbf{x}_2^{(2)}$ (connected with a dotted line) which form the local mean at

$\mathbf{m}_2(\mathbf{x}_1^{(1)})$. For $\mathbf{x}_2^{(1)}$ the two nearest neighbours are $\mathbf{x}_1^{(2)}$ and $\mathbf{x}_3^{(2)}$ which form the local mean at $\mathbf{m}_2(\mathbf{x}_2^{(1)})$. Similarly, local means can be derived from the same class for the calculation of within-class scatter.

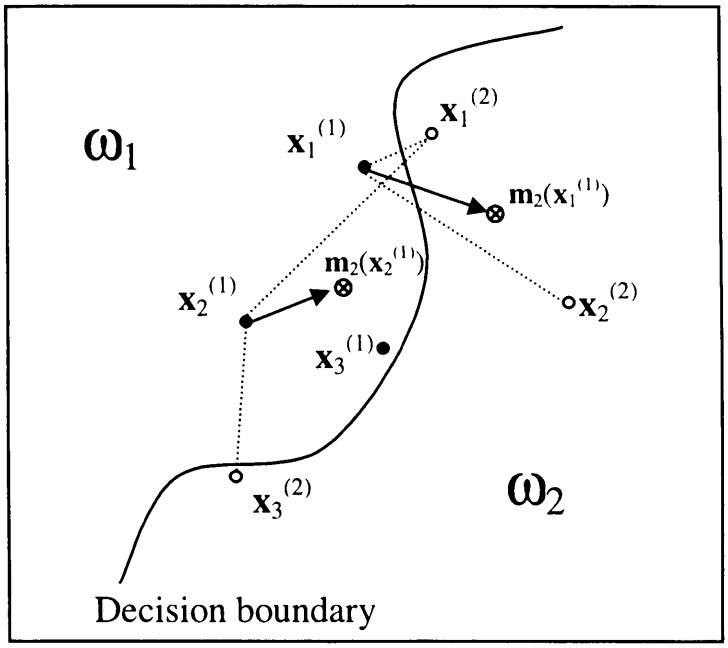


Figure 3-2. Illustration of local mean derivation for the calculation of between-class scatter with k=2. '•' for samples in class ω_1, '°' for samples in class ω_2, '⊗' for local means. The arrow indicates the direction from the sample to its local mean.

From $\mathbf{S}_b$ and $\mathbf{S}_w$ (equations 3-8 and 3-9 respectively), the separability measure between ω_1 and ω_2 is defined as:

$$J = \frac{1}{n}\mathrm{tr}(\mathbf{S}_w^{-1}\mathbf{S}_b) \tag{3-11}$$

Note that to ensure that J = 1 when the two classes are identical, the trace of $\mathbf{S}_w^{-1}\mathbf{S}_b$ is divided by dimensionality n. If J is close to one or even smaller than one, the two classes have low separability. The larger J is the higher the

separability between the two classes. A very large J indicates no overlap between classes.

For a multi-class problem, the values for J between all possible pairs of classes are mutually computed. This gives us a separability matrix. The main-diagonal entries of the matrix are 1, and it is symmetric (which is consistent with the definition of the separability measure formula).

3.4 Separability versus classification error

It is assumed that classes in a dataset with high separability should be easier to classify (Theodoridis & Koutroumbas, 1999). This implies that if the separability measure is workable, the classifier will obtain a lower classification error for problems with higher separability. This section employs a commonly-used classification problem in pattern classification research to evaluate the proposed separability measure.

Figure 3-3 shows the class distribution. Each of the four clusters (of two classes) can be expressed by the conditional probability density functions as follows.

$$f(\mathbf{x}|\omega_i) = \frac{1}{2\pi\sigma_i^2} \exp\left(-\frac{1}{2\sigma_i^2} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 \right) \qquad (3\text{-}12)$$

where $\boldsymbol{\mu}_i$ is the mean vector and σ_i^2 is the variance.

It is clear that the separability of the two classes cannot be effectively measured by classical separability measures (Section 3.2.1), because the class distributions are multimodal and share the same mean.

In this experiment, we change the mean vector in (3-12) in order to simulate datasets with different separabilities. Figure 3-3 shows two exemplar datasets with high separability and low separability.

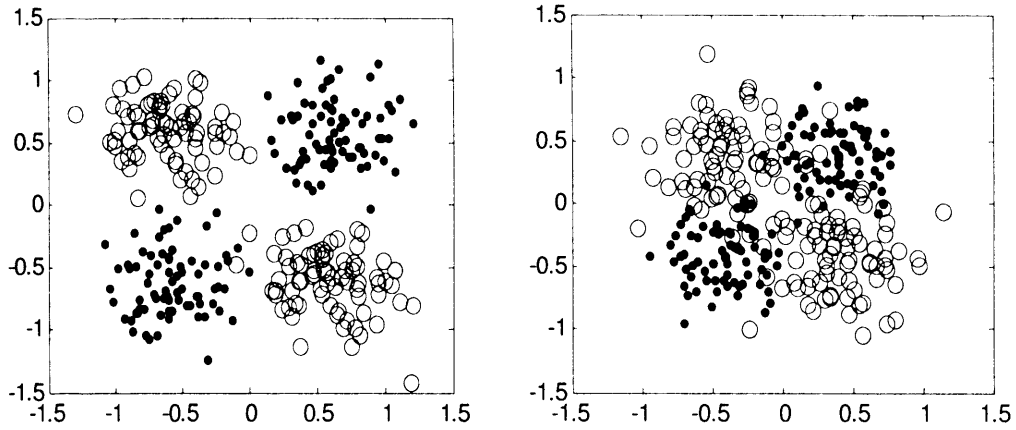


Figure 3-3. Distribution of a nonlinear classification problem with two classes. '•' - class 1, '°' - class 2.
(a) Higher separability between classes ω_1 and ω_2.
(b) Lower separability between classes ω_1 and ω_2.

A series of datasets with different separability were generated from (3-12). Each dataset contained 2000 samples, 1000 samples were used to train the classifier and the other 1000 samples were used to test the classifier performance. For each of the datasets, MLP and RBFN classifiers were designed. The structure of all classifiers were fixed, that is, both MLP and RBFN had two input nodes for the

two-dimensional vector, and two output nodes for the two classes. The number of hidden nodes was set to 6 for MLPs and 30 for RBFNs.

The performance of the trained classifiers was evaluated on the test datasets. The experimental results are shown in Figure 3-4.

The results in Figure 3-4 demonstrate that the classification error decreases with the increase in separability. Therefore, on the basis of this study, it seems that the proposed separability can effectively predict the classification difficulty of the data.

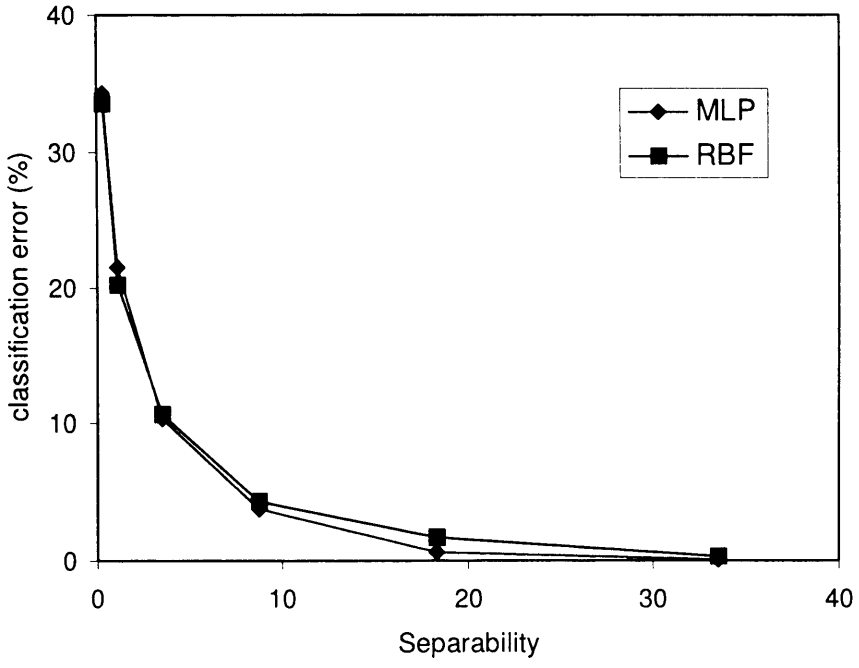


Figure 3-4. Separability versus classification error (%).

3.5 Conclusions

In this chapter, a separability matrix was presented that is suitable for use in the development of embedded CMFD applications. The separability matrix was derived from non-parametric analysis between classes in the data: it therefore requires no assumption about the underlying distribution of the data. In an assessment using a simulated classification problem, it was demonstrated that - in the case of both the MLP and RBFN classifiers considered in this thesis - the separability measure was able to predict the classifier performance.

The next chapter will use the proposed separability to develop a procedure for selecting the most appropriate pre-processing technique from possible candidate techniques for a given CMFD application.

4
IDENTIFYING THE MOST APPROPRIATE PRE-PROCESSING STRATEGY USING A SEPARABILITY MEASURE

4.1 Introduction

The effectiveness and reliability of a pre-processing technique has traditionally been assessed based on trial-and-error (Ribbens & Bieser, 1995; Grimmelius, *et al*, 1999; Staszewski, 2000; Yang, *et al*, 2002; Wang & Too, 2002) as follows:

1) Pre-process the data using different signal processing techniques to obtain sets of feature vectors;
2) Train individual neural networks using these feature vector sets;
3) Employ the pre-processing technique which results in the best overall classification performance.

Figure 4-1(a) depicts this traditional pre-processing selection procedure. From the figure, it is clear that this selection procedure requires the design of a classifier for each of the extracted feature vector sets. Since training of neural networks often takes a long time (due to the need to optimise training parameters and avoid local minima), designing different networks for each set of pre-processed data can be a laborious process (Ripley, 1995; Somol & Pudil, 2002; Marzi, 2002). In order to

avoid designing multiple classifiers and hence to improve selection efficiency, this chapter presents a procedure for identifying the most appropriate pre-processing techniques for a given set of recorded signals. This procedure is based on the non-parametric separability matrix proposed in the last chapter. The procedure is then assessed using a real CMFD problem of engine misfire detection.

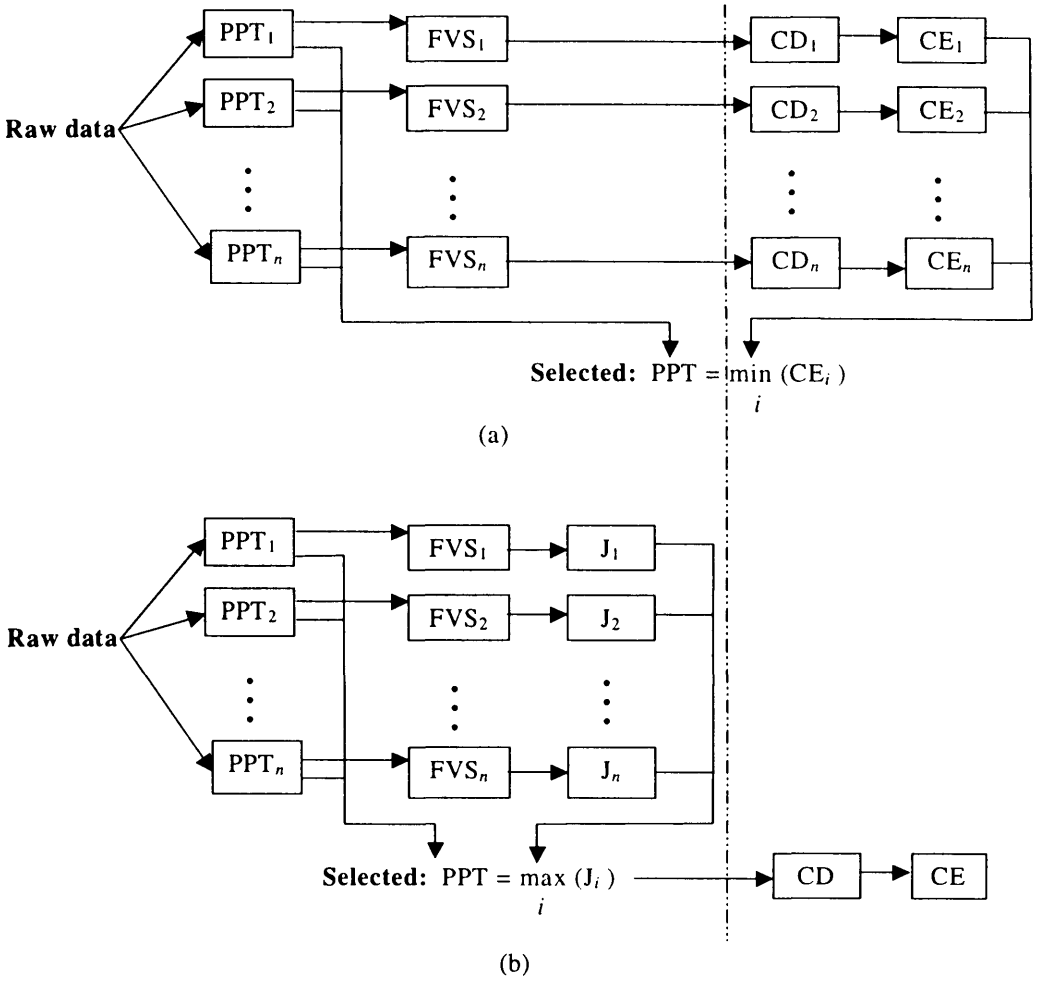


Figure 4-1. The pre-processing technique selection procedure.
(a) Traditional trial-and-error selection procedure.
(b) The selection procedure based non-parametric separability measure.
Keys to the figure: PPT (pre-processing technique), FVS (feature vector set), CD (classifier design), CE (classification error), J (separability measure).

4.2 Pre-processing technique selection based on separability measures

A procedure for selecting the most appropriate pre-processing technique for a particular classification problem is presented as follows.

1) Take the recorded raw dataset and apply the chosen pre-processing technique.
2) Measure the separability between classes after pre-processing.
3) Repeat for all alternative pre-processing techniques.
4) In the classifier system, employ the pre-processing technique that results in the largest separability measure.

This selection procedure is illustrated in Figure 4-1(b). Comparing it to the traditional selection procedure as in Figure 4-1(a), it is only necessary to design a single classifier for the feature vector set with the highest separability.

To assess this approach, the remainder of this chapter will apply it to a real CMFD problem involving engine misfire detection. Before conducting the assessment, the next section first provides some background to the topic of misfire detection, and describes the procedure for data acquisition.

4.3 Misfire detection: background and data acquisition

4.3.1 Background

Misfire in a petrol engine is a condition in which there is no combustion of the fuel/air mixture during the power stroke of the engine (Ribbens, *et al*, 1994). When misfire occurs, engine performance suffers, along with fuel economy and idle quality.

Of particular concern is the fact that, during misfire, there is an increase in the level of exhaust emissions. As a result, misfire is one of the key areas of concern in On-Board Diagnostics generation 2 (OBD II). OBD II is a collection of strict emissions oriented monitoring rules for US passenger cars (Carley, 1997). Similar regulations are expected in Europe.

Engine misfire detection has been extensively investigated in the last decade. This investigation has resulted in a number of detection methods and a number of publications (Wu & Lee, 1998). No matter what the detection method is, the necessary task is to obtain misfire information from the recorded signals, such as rotational speed (Williams, 1996), exhaust pressure (Ceccarani, *et al*, 1998), sound (Li, *et al*, 1996; Li Z, *et al*, 1997) or vibration. It is rare that the recorded signals can be directly used for misfire detection, rather they must be pre-processed using signal-processing techniques. The use of a certain signal processing technique has depended on the investigator's preference (Staszewski, 2000; Liu, *et al*, 2002). Therefore the selected signal processing technique is not

necessarily the most appropriate. While this chapter is concerned with the development of a method for selecting effective pre-processing techniques, this makes the misfire detection problem particularly valuable as the case study to assess the proposed method.

4.3.2 Data acquisition description

In the present experiment, vibration data were used to detect the misfire situation.

The studies involved a 6-cylinder Ford 2900 spark ignition (SI) engine. The engine was installed in the engine test cell in the Department of Engineering, University of Leicester and a water-brake dynamometer was connected to apply a load.

Three accelerometers (along the three axes) were mounted to the engine block via an adapter. An encoder on the crankshaft was used for the timing mark. The transducer signals were conditioned and were recorded digitally on line at a 10kHz sampling rate.

The experiments were concerned with sustained misfires, introduced by disconnecting the lead of cylinder three in the engine. The engine was run at 2700 revolutions per minute with varying loads of 50Nm, 60Nm, 70Nm, 80Nm, 90Nm and 100Nm respectively for both normal and misfire conditions. The recorded signals were then pre-processed to form feature vectors that were fed into the

classifier to give an indication of the engine firing state (that is, normal or misfire).

Figure 4-2 shows a schematic representation of the engine misfire detection system.

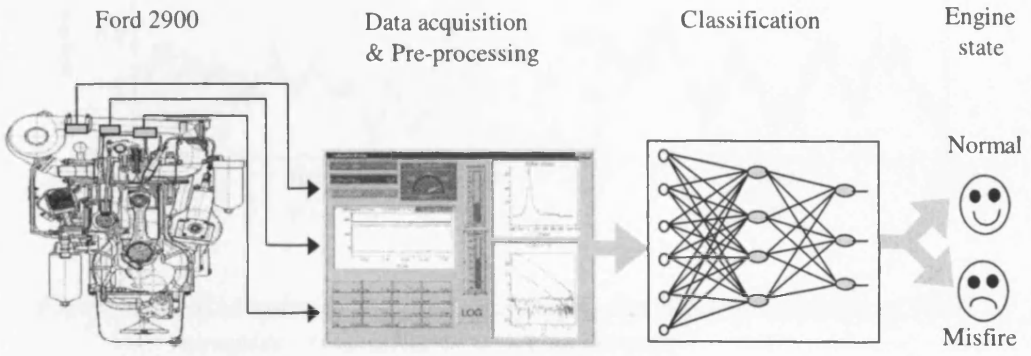


Figure 4-2. Schematic diagram of misfire detection system.

Figure 4-3 shows 2 normal and 2 misfire examples of the recorded vibration signals. Each signal has 1000 samples. Since the absolute amplitude of the signal has no effect on the results of misfire detection (as long as all the signals have the same scale), the y-axis has the default scale of recording, without calibration.

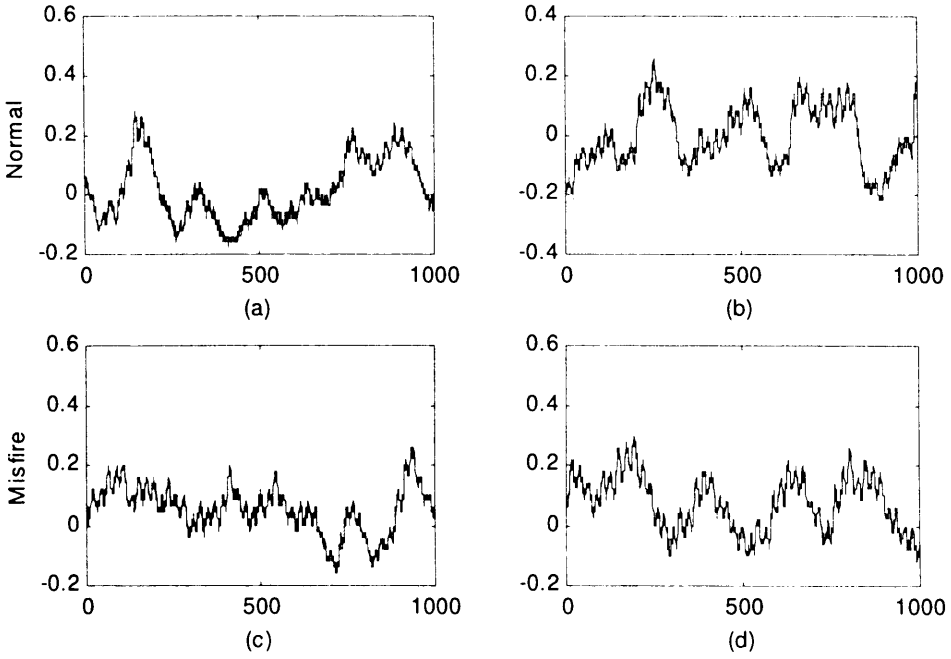


Figure 4-3. Examples of the logged signals. Each signal consists of 1000 samples, its amplitude is not calibrated.
(a)(b) Normal condition,
(c)(d) Continuous misfire in cylinder 3.

4.4 Feature vector sets after pre-processing

As in many real CMFD examples, the 'raw' vibration signals (each containing 1000 samples) were rather too large to classify directly with a neural classifier.

Three pre-processing schemes were therefore considered and compared using the separability matrix described in Chapter 3. To make the number of samples equal to a power of 2 (due to signal processing restrictions), 512 samples of the 1000 samples (for each signal, along two axes) were used in the subsequent feature extraction.

4.4.1 Features from time domain

The first pre-processing technique employed involved 'down sampling' the original (time domain) data. This 'down sampling' is explained in this section.

Let the engine speed be n revolutions per minute (r/min), that is, the engine rotation frequency is $n/60$. For four-stroke-cycle engines, each of the cylinders fires once in any two successive revolutions. For m cylinders the firing frequency is therefore: $f_f = n \cdot m / 120\text{Hz}$. In this study, the engine has six cylinders ($m = 6$) and the original signal sampling rate was 10kHz. For these tests, the engine was made to run at approximately 2700 r/min (n). Thus, f_f = 135 Hz, this gives 74 (10k/135) samples between two cylinders firing. Assuming that at least two samples between cylinders firing in the down-sampled data are required (according to sampling theorem), only 1 sample point in 37 samples needs to be retained. For convenience, and to ensure consistency with other techniques discussed below, one point from every 32 samples was selected. Therefore 16 data points were used to represent 512 samples: each such set of 16 points was considered as a feature vector (a pattern) for one signal. Data from two axes were used in each case. In this way, 360 'time domain' 32-dimension feature vectors (that is, 180 for normal and 180 for misfire) were produced.

4.4.2 Features from frequency domain

Power spectrum estimation is a frequency analysis technique widely used for the processing of vibration signals (Tandon & Choudhury, 1999). For a time signal, denoted by $x(t)$, the Fourier transform $S(f)$ is given by:

$$S(f) = \int_{-\infty}^{\infty} x(t)\mathrm{e}^{-j2\pi ft}\mathrm{d}t = |S(f)|e^{j\theta(f)}$$

where $|S(f)|$ and $\theta(f)$ represent the amplitude and phase of the Fourier transform, respectively.

The power spectrum, $P(f)$, of $x(t)$ is defined as (Newland, 1993):

$$P(f) = E\left(S(f)S^{*}(f)\right) = E\left(|S(f)|^{2}\right)$$

where * denotes the complex conjugate.

For the recorded vibration signals (each with 512 samples), the power spectra were estimated using a Hanning window, without overlapping (Oppenheim, 1975). By investigating the power spectra, it was observed that the signal power is contained in 16 frequency components. Thus 16 points of the most significant information components of each power spectrum were chosen to form a feature vector.

As with the time-domain signals, with the result that data from two axes were used in each case, 32-dimension feature vectors were produced.

4.4.3 Features from wavelets coefficients

Wavelet analysis is used to decompose a time-domain signal into a series of wavelets at different levels. Each of the wavelets in the time domain has the same length as the original signal, but covers a different frequency band. By selecting

and examining one or more of these wavelets, one can derive the desired information and remove unwanted parts from the original signal.

For an orthogonal wavelet transform, a signal $x(t)$ at $t \in [0,T]$ can be decomposed into a summation of wavelets at a finite number of scales/levels (Staszewski, 1998) as:

$$x(t) = a_0 + \sum_{j=0}^{\infty} \sum_{k=0}^{2^j-1} a_{2^j+k} w(2^j t - k) \qquad (4\text{-}13)$$

where a_i are coefficients of the wavelet transform, $w(2^j \cdots)$ are wavelets of level j. In this application, the twentieth-order Daubechies' wavelet is used to decompose the vibration signal. Figure 4-4 shows an example of the vibration signal and its corresponding wavelet transform. Since the signal does not need to be calibrated for the purpose of misfire detection, the units of amplitude were not labelled in the figure.

Eight levels of wavelet decomposition for the vibration signal in Figure 4-4 are shown in Figure 4-5. The spectral for corresponding wavelet levels in Figure 4-5 are shown in Figure 4-6. It can be seen that each of the wavelet levels covers a different frequency band.

In this experiment, the engine was run at 2700r/min and the firing frequency was approximately 135Hz. By analysing the spectra of wavelet levels in Figure 4-6, this firing frequency was found to dominate the fourth level of wavelet

decomposition. So the 16 wavelet coefficients representing the fourth wavelet level were used for misfire detection.

Data from two axes were used in each case, so 180 normal and 180 misfire 32-dimension feature vectors were created in this way.

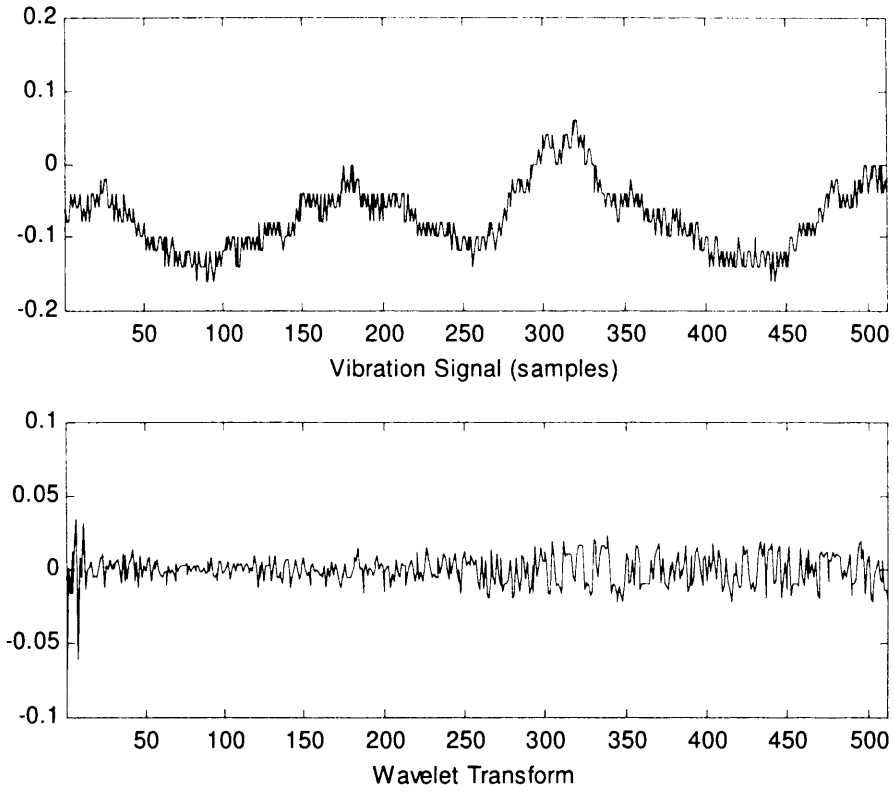


Figure 4-4. Vibration signal and Wavelet transform.

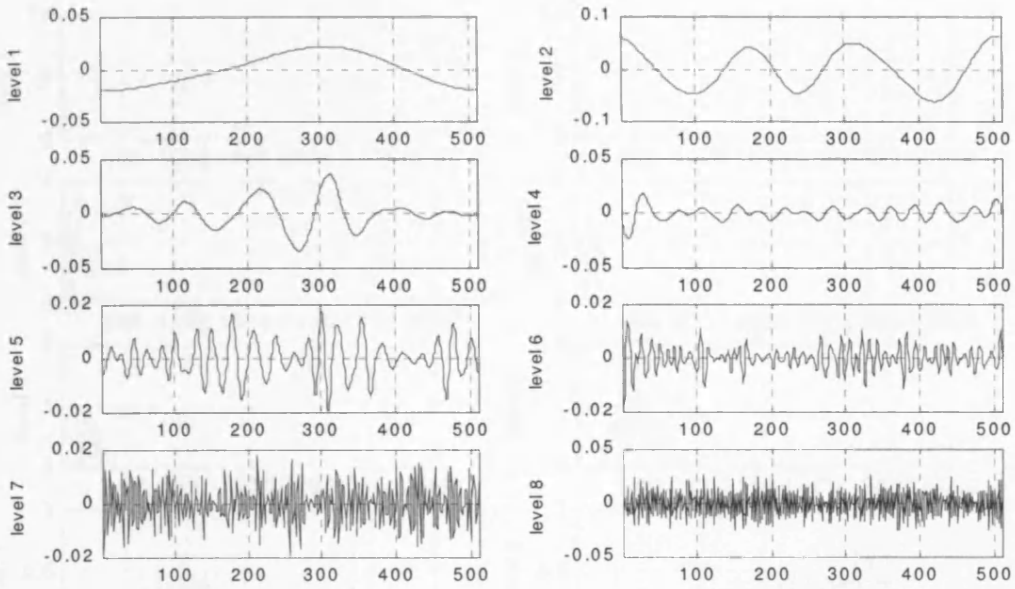


Figure 4-5. Wavelet decomposition.

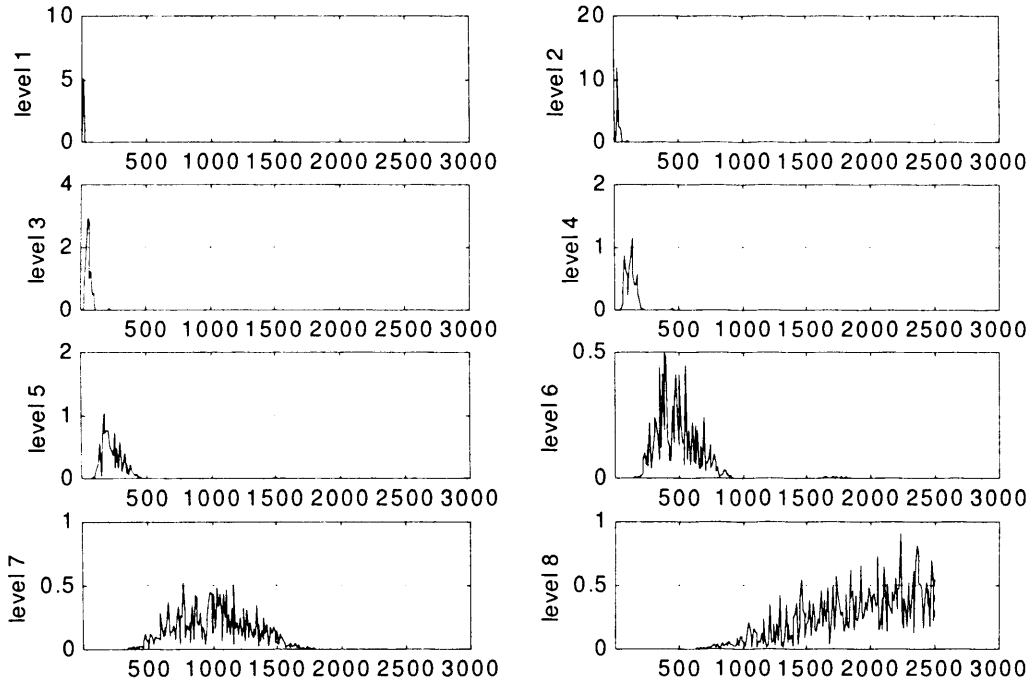


Figure 4-6. Spectral of wavelet levels (Horizontal axis unit is Hz).

4.5 Prediction of pre-processing efficiency

The intention of this study was to explore the extent to which the separability matrix assists in the selection of the most appropriate pre-processing technique.

The separability between normal and misfire conditions for the time domain dataset was calculated using (3-11), and is listed in Table 4-1. Similarly, the frequency domain results are given in Table 4-2, and the results for the wavelet transform are in Table 4-3.

	Normal	Misfire
Normal	1	1.14
Misfire	1.14	1

Table 4-1. Separability matrix for the engine misfire dataset (Time Domain).

	Normal	Misfire
Normal	1	1.18
Misfire	1.18	1

Table 4-2. Separability matrix for the engine misfire dataset (Power Spectrum).

	Normal	Misfire
Normal	1	2.84
Misfire	2.84	1

Table 4-3. Separability matrix for the engine misfire dataset (Wavelet).

It is observed that the separability of the dataset from wavelet coefficients is the biggest, while the time domain is the smallest. According to the discussion in Section 4.2, the dataset with the greatest separability should result in the most effective classification performance.

In the next section we explore the predictive value of this technique.

4.6 Implementing the classifiers

As mentioned above, three datasets, each contains 180 normal and 180 misfire 32-dimension feature vectors for misfire detection were produced from the time domain, power spectrum and wavelet analysis respectively.

In estimating the classification errors of the designed classifiers on datasets from different pre-processing techniques, each of the three datasets was equally partitioned into a training set and a test set. That is, the training set contained 90 normal and 90 misfire feature vectors, and the test set contained 90 normal and 90 misfire feature vectors.

All classifiers had 32 input nodes for the 32 dimensions of the feature vector, and 2 output nodes for representing the two engine conditions (normal and misfire). The architecture of the classifiers was 32-*M*-2, where *M* is the number of hidden nodes of the MLP, or the number of radial basis functions of the RBFN.

For the MLP, M=40 was found to produce the best classification result after several trials. Therefore, in this experiment, an architecture of 32-40-2 for MLPs was used for engine misfire detection.

For the RBFN, the number of hidden neurons of the classifiers for time domain, power spectrum and wavelet coefficients was set to 80, as these values were found to produce the best results.

Table 4-4 lists the classification error of the classifiers from features extracted from the time domain, power spectrum and wavelet coefficients

	MLP	RBFN
Time domain	35.6	34.4
Power spectrum	34.4	33.3
Wavelet coefficients	13.9	15.6

Table 4-4. Misfire detection error (%) using different pre-processing strategies.

As discussed in Section 4.5, the separability between normal and misfire condition is 1.14, 1.18 and 2.84 in the feature space formed from time domain, power spectrum and wavelet coefficients respectively.

From the results in Table 4-4, it is observed that, for both MLP and RBFN, the classification error is the smallest on the dataset derived from wavelet coefficients and the largest on the dataset derived from the time domain. This negatively correlates with the separabilities for the datasets. Overall, this experiment suggests that the proposed method is effective in selecting an appropriate pre-processing technique for fault classification applications.

4.7 Further experiments

In the last section it was demonstrated that the best pre-processing techniques could be selected based on values of the proposed separability measure. This section carries out an experiment to further validate the claim made. The experiment is concerned with whether the selection strategy is dependent on the separability values only. That is, for the same set of recorded signals, the pre-processing technique obtaining largest separability value should result in the lowest classification error. This further study is particularly useful because we may have some variations in experiment set-up: for example, we may record different sets of signals. Therefore the purpose of this further experiment is to validate the claim: given a set of recorded signals, the proposed selection method should be capable of identifying the most appropriate pre-processing technique.

In justifying this argument, a sub-set of the whole set of signals (as used in the last section) was formed. From the sub-set signals, three sets of feature vector were extracted from time domain, power spectrum and wavelet coefficients respectively. Each set contained 200 feature vectors, that is, 100 vectors of normal condition and 100 vectors of misfire condition. The separability was calculated for each of the feature sets, and listed in Table 4-5[5].

	Separability between Normal and Misfire
Time domain	1.49
Power spectrum	1.63
Wavelet coefficients	2.23

Table 4-5. Separability for feature vectors from the sub-set of signals.

Similarly each set was equally partitioned into a training set and a test set. Table 4-6 lists the classification errors for feature vectors extracted from this sub-set of signals.

[5] It is observed that the value of the separability of this dataset differs from that computed on the dataset from the whole set of signals. This is plausible because the sub-set and the whole-set have different distributions. Here we are not concerned with the difference in the distributions, we simply focus on how to identify the most appropriate pre-processing technique for a given set of signals.

	MLP	RBFN
Time domain	28	29
Power spectrum	27	26
Wavelet coefficients	18	20

Table 4-6. Classification error (%) on the feature sets from the sub-set of signals.

By examining the results listed in Table 4-5 and Table 4-6, it is clear – again - that the classification error decreases with the increase of separability.

4.8 Summary

The two experiments described in this chapter were conducted to assess the claim: given a set of signals, the proposed selection procedure is able to identify the most appropriate pre-processing technique.

Figure 4-7 summarises the results from the two experiments. From this figure, it is clear that, for both MLP and RBFN classifiers, the classification error strongly correlates with the separability.

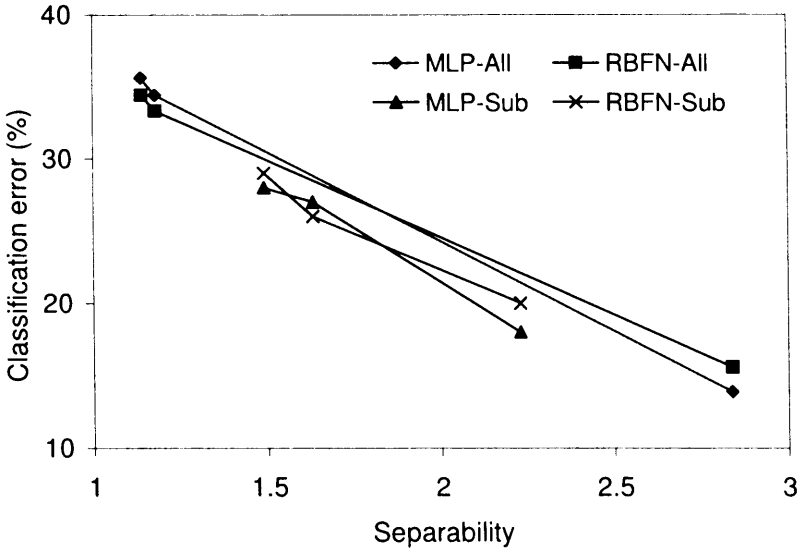


Figure 4-7. A summary of the results on misfire detection using the whole-set or a sub-set of the recorded signals. 'All' indicates the whole set of signals. 'Sub' indicates the sub-set of signals.

4.9 Conclusions

This chapter proposed an efficient selection procedure which was based on the separability analysis of feature vector sets. The proposed procedure selects the pre-processing technique which results in the feature vector set with the highest level of class separability. In the assessment using a real CMFD problem, it was demonstrated that - in the case of both the MLP and RBFN classifiers considered in this thesis - the selection procedure was able to identify the most appropriate pre-processing technique.

The next chapter will move on to the second stage of developing embedded CMFD systems: classifier design.

5
CLASSIFIER COMPARISON CRITERIA AND CASE STUDIES

5.1 Introduction

As noted in Chapter 1, this thesis is concerned with the design of three-stage CMFD systems. In such systems, after the recorded signals have been pre-processed, the processed signals must be classified. The performance of the classifier thus plays a central role in the whole CMFD process.

A classifier may be suitable for some kinds of problem and not suitable for others, depending on the used performance measuring criteria (Michie, 1994; Mak, *et al*, 1993; Wilson, *et al*, 1997; Terra & Tinos, 2001; Liu & Gader, 2002; Prakash, *et al*, 2002). Thus an appropriate classifier needs to be selected early in the design process.

The effectiveness of neural network classifiers has traditionally been compared using empirical studies (Zhang, 2000; Prechelt, 1996). This chapter, and the two that follow, are devoted to such an empirical comparison, with a focus on practical CMFD applications.

This chapter provides the necessary background material for the experimental comparisons in the next two chapters, by:

1) identifying new comparison criteria based on the distinct features of embedded CMFD applications;
2) describing datasets from classification problems with different levels of classification difficulty;
3) describing the results of experiments in which the basic classification performance of MLP and RBFN classifiers was compared using the above datasets.

5.2 The comparison criteria

To evaluate the performance of neural classifiers, a number of comparison studies have been carried out since 1990 (Michie, *et al*, 1994; Ripley, 1994; Jain & Mao, 1997; Zhang, 2000; Hsu & Lin, 2002). Notably, in 1997, IEEE Transactions on Neural Networks published a special issue on artificial neural networks and statistical pattern recognition techniques. In this issue, Holmstrom *et al* (1997) compared 18 statistical and neural classifiers on two datasets: hand-written digits data and phoneme data. The neural classifiers compared in their paper were multilayer perceptron and learning vector quantisation. It was concluded that, compared to statistical classifiers, the neural classifiers provide an attractive alternative by combining good classification performance and less complex design. Other earlier efforts in comparing neural classifiers date back to 1990. One of the first contributions was made by Cheng & Titterington (1994). Cheng & Titterington explored the links between neural networks and statistical

methodology. They showed that some statistical procedures can be given a neural network expression, and that neural networks can be provided with a statistical explanation/commentary. In the same year, Ripley (1994) published a paper setting up a framework for comparing classifiers from statistics, neural networks and machine learning. He compared Bayes' rule, linear discriminant analysis, logistic discriminant analysis, quadratic discriminant analysis, k nearest neighbours, multivariate adaptive regression spline, projection pursuit regression, classification tree, learning vector quantisation and MLP on three datasets: synthetic data, sonar data, and forensic glass data. It is also worth mentioning the project StatLog under the ESPRIT programme of the European Community (Michie, 1994). StatLog compared and evaluated 23 different classifiers from statistics, machine learning and neural networks on 22 datasets. The results showed that there was no unique best classifier in terms of classification accuracy. On analysing published results in comparison, Duin (1996) argued that a straightforward and fair comparison should be carried out in a defined application domain with good comparison criteria.

Chapter 1 analysed the distinct characteristics of CMFD applications. From the discussions in that chapter it is clear that, in addition to the evaluation of classification accuracy, the criteria for CMFD applications must include the assessment of the classifier's ability to deal with multiple faults, unknown faults and limited amounts of available data.

In addition, as also discussed in Chapter 1, embedded systems have - compared to 'desktop' and 'workstation' computers - severely limited memory and CPU power. Therefore, in contrast to the great majority of previous studies in this area, the hardware resources required to implement MLP and RBFN classifiers will be considered in some detail in this study. Hardware requirements must be assessed in two ways: processor requirements and memory requirements. Here the former requirement refers to the processor time needed to train and apply the classifier, while the latter requirement indicates how much memory is required to implement a classifier.

In summary, the criteria to be used in this thesis for the comparison of MLP and RBFN classifiers for use in embedded CMFD applications are as follows:

- Basic classifier performance in terms of classification error (or classification accuracy).
- The ability to detect unknown faults.
- The ability to deal with multiple faults.
- The effects of dataset size on generalisation ability.
- The processor requirements.
- The memory requirements.

5.3 The case studies

When examining the literature, the history of neural classifier comparison can be roughly divided into two periods: pre 1996 and post 1996. The research on neural classifier comparison began around 1990 and attracted increasing interest in the

following years (Michie, 1994; Blayo, *et al*, 1995; Prechelt, 1994; Zheng, 1993). In the early period, although researchers realised that classifier performance can only be compared using different application problems, many published works used no more than one problem in their studies [6]. Using a single problem for neural network comparisons is now considered insufficient, because a single problem cannot represent the variety of classification difficulties. Since the publication of some distinguished researchers' work (for example, Michie, 1994; Prechelt, 1996; Duin, 1996) in which they strongly suggested that editors and reviewers should set significantly higher standards, most researchers have used multiple problems to evaluate classifiers. In line with these recommendations, the present study involves data from multiple problems.

Having decided to base the comparison on data from multiple problems, the next question is what kind of problems should be employed. Generally classification problems can be differentiated into artificial, realistic and real problems (Ripley, 1994; Prechelt, 1996). Prechelt (1996) described these three kinds of problems as follows:

---

[6] Prechelt (1996) analysed articles published in 1993 and 1994 in four of the oldest and most well known journals dedicated to neural network research. The four journals are Neural Networks, Neural Computation, Neurocomputing, and IEEE Transactions on Neural Networks. He observed that around 40% of the articles used no more than one problem. He suggested that in future articles not using a minimum of two problems "should usually be rejected". This view was echoed by researchers in experimental evaluation of neural networks. Since then using multiple problems was becoming the standard practice in experimental evaluation of neural networks.

- Artificial problems are those whose data are generated synthetically based on some simple logic or arithmetic formula, for example, encoder/decoder, sine wave, etc.
- Realistic problems also consist of synthetic data, but these are generated by a model with properties similar to those can be found in the physical world.
- Real problems consist of data that represents actual observations of phenomena in the physical world.

Although artificial problems are weak in connection with the real world, they are commonly acceptable and have served for an illustration of a classifier performance in many publications. Realistic problems are considered useful to assess the behaviour of a classifier on problems with known properties, they provide the best way to characterise the kinds of problems for which a classifier will yield good results. Real problems usually have characteristics that are not completely known, yet they act as a real challenge for classifier performance.

This study uses the following classification problems:

1) A mathematical model (described in Section 5.3.1);
2) A non-linear cooling system model (described in Section 5.3.2)
3) A breast cancer diagnosis dataset (described in Section 5.3.3).

Problem 1 is a realistic problem because the model of this type is quite general and fits, for example, the linearlised engine model of Hsu *et al* (1995). Problem 2 is a complex realistic problem. Problem 3 is a real problem, it is publicly

available and has been used in many classifier comparison research (for example, Lampariello & Sciandrone, 2001; Heinke & Hamker, 1998).

Each problem is described, in turn, in the following sub-sections.

5.3.1 The Mathematical Model (MM) case study

The following mathematical model represents a large class of static diagnosis problems and is adapted from that introduced by Kramer & Leonard (1990), and described as follows:

$$\mathbf{Y} = \mathbf{Y}_0 + f(\mathbf{p}) + \mathbf{v} \tag{5-14}$$

Here $\mathbf{Y}$ represents the measurement vector of the plant, $\mathbf{Y}_0$ represents the plant nominal steady state. Measurement $\mathbf{Y}$ is a function of plant physical parameters $\mathbf{p}$, $f(\mathbf{p})$, and suffers from measurement noise $\mathbf{v}$.

Plant faults are caused by deviation of parameters. All parameters are scaled such that their numerical value is zero at the nominal operating point. To simplify the problem, $\mathbf{Y}$ is assumed to be a linear function of $\mathbf{p}$, that is:

$$\mathbf{Y} = \mathbf{Y}_0 + \alpha\mathbf{p} + \mathbf{v} \tag{5-15}$$

α is a distribution matrix of parameter effects on the measurement vector. Here we assume that $\mathbf{Y}$ has two measurements y_1 and y_2, $\mathbf{p}$ has two parameters p_1 and p_2, and:

$$\alpha = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

This means that fault p_1 causes y_1 and y_2 to deviate in the same direction, and fault p_2 causes y_1 and y_2 to deviate in the opposite direction.

The classes are defined as:

Normal (C_0): $|p_1| < 0.05, \quad |p_2| < 0.05$

Fault 1 (C_1): $|p_1| > 0.05, \quad |p_2| < 0.05$

Fault 2 (C_2): $|p_1| < 0.05, \quad |p_2| > 0.05$

$v_1, v_2 \sim N(0,\ 0.015)$

One set of training data was generated with values of p_1 and p_2 sampled from the normal distribution $N(0,\ 0.25)$. In total, 300 input/output pairs (*i e.* 100 input/output pairs per class respectively) were generated from (5-15) and were used for training all networks.

Figure 5-1 illustrates the class distribution in the measurement space.

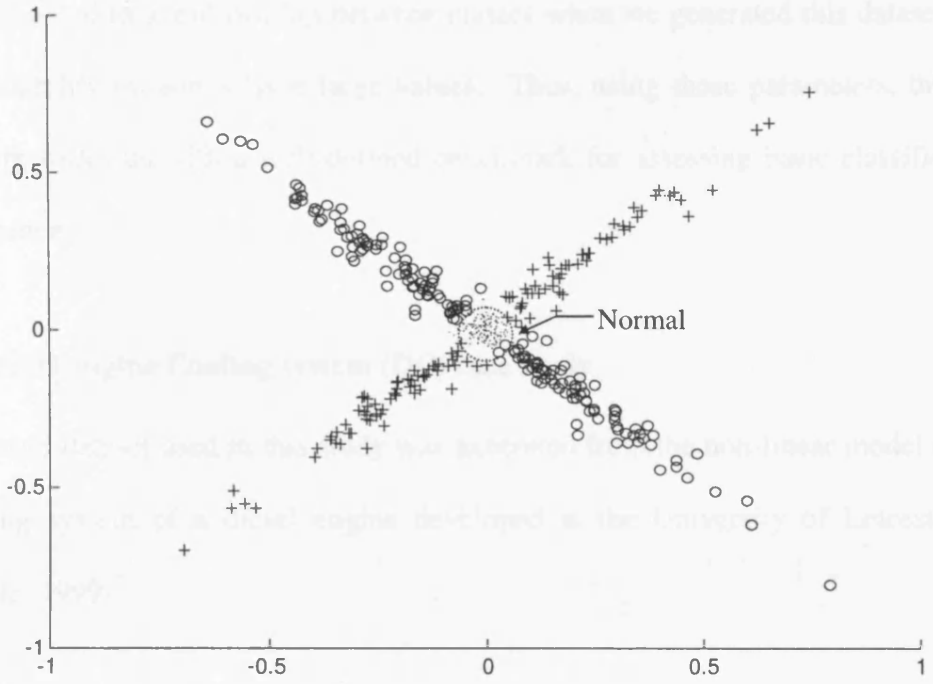


Figure 5-1. Class distribution of mathematical model dataset '.'--Normal, '+'--fault 1, 'o'--fault 2.

One additional set of test data was also generated. This dataset was intended to explore the generalisation (interpolation) ability of the networks and had the same distribution as the training set.

Table 5-1 shows the separability matrix (using k = 1 in the separability measure proposed in Chapter 3) for a combination of the training and testing datasets.

	Normal	Fault 1	Fault 2
Normal	1	141.88	85.42
Fault 1	141.88	1	180.37
Fault 2	85.42	180.37	1

Table 5-1. Separability Matrix for the 'Mathematical Model' Dataset.

Since we opted to avoid overlap between classes when we generated this dataset, the separability measures have large values. Thus, using these parameters, this dataset provides us with a well-defined benchmark for assessing basic classifier performance.

5.3.2 Diesel engine Cooling system (DC) case study

The second dataset used in this study was generated from the non-linear model of a cooling system of a diesel engine developed at the University of Leicester (Twiddle, 1999)[7].

Such a cooling system is designed to rapidly bring the engine to its most efficient operating temperature and to maintain this temperature even when (for example) the load and speed are varied. This section briefly describes the non-linear thermodynamic cooling system model which was used to generate data for fault diagnosis of a diesel engine cooling system. A full description of the model is available elsewhere (Twiddle, 1999).

The cooling system may be considered as two models, a model for heat transfer from the engine block to coolant, and a model for heat dissipation from the radiator.

---

[7] Special thanks to Dr John Twiddle who kindly provided the model and the data for this study.

For the model of heat transfer from the engine block to coolant:

$$\frac{dT_2}{dt} = \frac{h_b A_b}{m_b C_c}(T_b - T_2) - \frac{\dot{m}_b C_c}{m_b C_c}(T_2 - T_1)$$

where: h_b is the convective heat transfer coefficient between the block and the coolant.

A_b is the internal area of contact between the block and the coolant over which this heat transfer takes place.

m_b is the mass of coolant contained within the block.

C_c is the specific heat capacity of the coolant.

T_b is the engine block temperature

T_1 is the coolant temperature at engine block inlet.

T_2 is the coolant temperature at the engine block outlet.

For the model of heat dissipation from the radiator:

$$\frac{dT_3}{dt} = \frac{\dot{m}_{rad} C_c}{m_{rad} C_c}(T_2 - T_3) - \frac{h_{rad} A_{rad}}{m_{rad} C_c}(T_3 - T_a) - \Sigma\sigma F A_{rad}(T_3^4 - T_a^4)$$

where: $m_{rad} C_c$ is the mass of coolant in the radiator multiplied by the specific heat capacity, C_c, of the coolant.

$h_{rad} A_{rad}$ is the radiator heat transfer coefficient multiplied by the total surface area of the radiator.

σ is Boltzmans constant

Σ is relative emissivity for the surface of the radiator

F is defined as the shape factor

T_a is the ambient temperature

T_3 is the coolant temperature at the radiator outlet

In the above equations the mass flow rate of coolant is assumed to be proportional to rpm, N, and is given by:

$$\dot{m} = k_{pump} N$$

Using the model, various faults may be simulated, including those considered in this study: 'fan fault' (that is, the radiator fan is permanently off), 'thermostat fault' (that is, the thermostat is stuck open) and 'pump fault' (the coolant pump is damaged). To detect these faults, we have access to six measurements: the ambient temperature (T_a); the engine block temperature (T_b), the coolant temperature (T_1) at engine block inlet; the coolant temperature (T_2) at the engine block outlet; the coolant temperature (T_3) at the radiator outlet; and the engine speed (N).

Using the model, a training dataset (with 300 samples) and a testing dataset (with 300 different samples) were created. In each case, the datasets consisted of equal numbers of 'normal', 'fan fault', 'thermostat fault', and 'pump fault' data (75 samples of each).

The separability matrix (again with k = 1) for a combination of both testing and training datasets is shown in Table 5-2. Note in particular the fact that the 'normal' and 'thermostat fault' classes overlap in these datasets. This represents the fact that, with the available measurements, it is not always possible to distinguish between these two situations. As a result, this dataset is particularly valuable in this study as an example of 'overlapping classes'.

	Normal	Fan fault	Thermostat fault	Pump fault
Normal	1	268.54	0.67	56.45
Fan fault	268.54	1	244.61	306.35
Thermostat fault	0.67	244.61	1	62.45
Pump fault	56.45	306.35	62.45	1

Table 5-2. Separability Matrix for the Cooling System Dataset.

5.3.3 Breast Cancer (BC) case study

A publicly accessible breast-cancer dataset was also used in this study[8]. Originally this breast cancer dataset was obtained from the University of Wisconsin Hospitals, Madison (Mangasarian & Wolberg, 1990).

Here the classification task involves distinguishing between datasets derived from cancer cells (Heinke & Hamker, 1998). The dataset is characterised by small overlaps and complex decision boundaries. There are two pattern classes: benign tumour and malignant tumour. The two classes are distinguished by nine parameters. These include the clump thickness, the uniformity of cell size and cell shapes, the amount of marginal adhesion, and the frequency of bare nuclei. These various values were collected by microscopic examination. There are 690

[8] This dataset was clearly not from the CMFD domain. An important reason why it was used here is that it is readily available for public use from the University of California. http://www.ics.uci.edu/~mlearn/MLRepository.html. As a result, it makes it possible for other researchers to use the results presented here to 'benchmark' their own classifiers.

examples in the original dataset: in this study, 600 examples were used. The data were then divided into a training set and a test set, each with 300 examples. In the training set, the class of benign tumour has 160 examples and the class of malignant tumour has 140 examples, note that we have attempted to keep the size of the two training classes roughly equal (see Parikh, *et al*, 1999, for a discussion of this issue). In the testing set, the class of benign tumour has 215 examples and the class of malignant tumour has 85 examples.

The separability matrix (again with k = 1) for breast cancer datasets is shown in Table 5-3.

	Benign tumour	Malignant tumour
Benign tumour	1	4.62
Malignant tumour	4.62	1

Table 5-3. Separability Matrix for the 'Breast Cancer' Dataset.

5.3.4 Summary of dataset organisation

Datasets were obtained from three classification problems: a mathematical model for static fault diagnosis, a non-linear cooling system model and breast cancer diagnosis. The datasets were characterised using the proposed non-parametric separability analysis method. These problems each pose a different challenge to classifiers. The dataset from the mathematical model has low dimensionality and good separability, this provides us with a well-defined benchmark for assessing

classifier basic performance. The dataset from the non-linear cooling system model has medium dimensionality and good separability between some classes and very low separability between others. This dataset is particularly valuable to investigate classifier performance in terms of overlapping classes. The breast cancer dataset is publicly available, it has small overlaps and a complex class boundary.

5.4 Basic classifier performance

In this section the basic classification performance of the MLP and RBFN classifiers is compared. In each case, datasets from three problems are employed: the mathematical model (MM) for some fault diagnosis problems, the diesel engine cooling system model (DC), and the breast cancer diagnosis (BC), as described in Section 5.3. The basic performance of a trained classifier is compared based on classification error rate on the testing dataset. This section begins with a discussion of the data for designing and testing the classifiers.

5.4.1 Data for designing and testing the classifiers

From the case studies described in Section 5.3, datasets for comparing classifier performance were organised. Approximately equal numbers of samples in all classes were assumed, that is, each of the classes has approximately equal *a priori* probability, P_i for $i = 1,2,\cdots c$. Although this assumption may not always be satisfied in some real applications, the unequal size of classes can be re-organised into groups of equal size. For techniques to accomplish this, Parikh *et al* (2000) make some useful suggestions.

From the mathematical model (MM), one set of training data was generated with values of p_1 and p_2 sampled from the output of normal distribution $N(0, 0.25)$. In total, 300 input/output pairs (that is, 100 input/output pairs per class respectively) were generated from (5-15) and were used for training all classifiers.

Two additional sets of test data were also generated, 'Test 1' and 'Test 2'. 'Test 1' had the same distribution as the training set. 'Test 2' had values distributed over the whole parameter space. For this set, the 'correct' results for patterns from regions out of the training set were determined by distance, that is, a pattern is assumed to belong to the nearest class. These datasets were used to explore the generalisation ability of each network, both in terms of interpolation (Test 1) and extrapolation (Test 2).

From the diesel engine cooling system (DC) model, both the training dataset and the test dataset had 300 samples with equal class sizes (equal class distribution probability). The number of samples was determined empirically, that is, further increasing the number of samples did not show a significant improvement in classifier performance.

For breast cancer (BC) diagnosis, the number of samples available was limited: as discussed in Chapter 1. Limited number of samples is a common problem in CMFD applications.

Table 6-1 lists class distribution probability and the number of samples in training and test datasets.

	Prior Class Probability	Training Set Samples	Test Set Samples	
MM	$\begin{bmatrix}0.33\\0.33\\0.33\end{bmatrix}$	300	Test 1 300	Test 2 300
DC	$\begin{bmatrix}0.25\\0.25\\0.25\\0.25\end{bmatrix}$	300	300	
BC	$\begin{bmatrix}0.625\\0.375\end{bmatrix}$	300	300	

Table 5-4. Number of samples in training and test datasets for each of the three experiments.

5.4.2 Experiments

The three experiments conducted to compare basic classifier performance are discussed below.

a) MM case study

Both MLP and RBFN classifiers for the MM study had 2 input neurons, 3 output neurons and M hidden neurons, represented by 2-M-3.

For the MLP, structures with 3, 4, 6 and 15 hidden neurons respectively were implemented and compared. For the RBFN, the maximum number of radial basis functions was set at 50, since the classification error was found to decrease very little after this number. Then the RBFN was trained with different spread constants for the Gaussian functions.

Table 5-5 summarises the performance of the two classifiers on this task. Note that the classification error is given as a percentage. The numbers in the second row of the table are the number of hidden nodes. Numbers in parentheses are the spread constants for the Gaussian functions used with the RBFN.

	MLP				RBFN			
Number of hidden neurons	3	***4***	6	15	50 (0.01)	***50 (0.025)***	50 (0.05)	50 (0.1)
Training error	33.7	***0***	0	0	2.3	***0.3***	0	0
Error for Test 1	39.7	***1***	0.3	0.3	5.0	***0.3***	0.3	0.3
Error for Test 2	33	***17.7***	26.3	27.7	4.67	***1.7***	18.67	10.3

Table 5-5. Classification error rate (%) for Static fault diagnosis.

From the table, the following observations can be made:

1) For this problem, the MLP with 4-hidden neurons and the RBFN with spread constant 0.025 provide the smallest classification error on the training set and testing sets. The errors (on the Test 1 dataset) are very similar for the two classifiers. Referring back to Table 5-1, it is clear that the classes have large separability, while the classification results here

have very small error rate for both MLP and RBFN. Thus the results illustrate that both classifiers perform well if there is a large separability between classes.

2) In considering these results, it should be noted that the MLP starts training from random initial weights and converges to a possible local minimum, while the RBFN converges to a global minimum if the output layer is linear and the positions of radial basis functions in the feature space are located optimally *a priori* (Looney, 1997). Thus, using an MLP, it is sometimes necessary to train more than once to obtain an 'optimum' result, a fact which may increase the training time substantially. In all of the tables presented in this chapter, the MLP training was performed five times or more to find the best performance. The results presented here are the best that were obtained over these runs. Table 5-6 gives the results of five runs of training for the MLP with 4 hidden neurons. It is clear from the table that each of the trainings may give a different classification error.

3) For practical applications, it can be the case that the training data do not fully represent the feature space. The resulting classification during 'testing' is then a result of either interpolation or extrapolation from the available (training) data.

   In terms of interpolation, it is observed that both classifiers exhibit very similar classification errors (0.3%) within the regions of the training set. On the other hand, in terms of extrapolation (on the Test 2 dataset), the classification error of RBFNs (1.7%) is better than that of MLPs (17.7%) (see Table 5-5) for samples from regions unrepresented by the training set.

This is a plausible result. The decision boundary of RBFN is formed based on distance and that of the MLP is unbounded. The MLP therefore 'arbitrarily' classifies unseen patterns which lie outside the regions described by the training set[9].

Runs	1	2	3	4	5
Error of training	1.0	1.0	1.0	17.0	7.3
Error for Test 1	1.0	1.0	1.0	19.3	12.0
Error for Test 2	39.7	39.7	38.0	23.0	24.3

Table 5-6. Results of MLP for 5 runs.

b) DC case study

The dataset used in the second study of basic classifier performance was generated from the non-linear model of a diesel engine cooling system described in Section 5.3.2. The classifier for diesel cooling system diagnosis is implemented as shown in Figure 5-2.

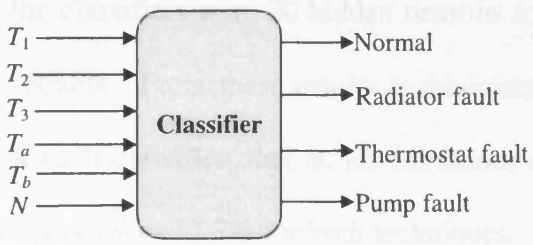


Figure 5-2. Classifier for Cooling system diagnosis.

[9] This is discussed in greater detail in Chapter 7.

In this experiment, both MLP and RBFN classifiers had 6 input neurons, 4 output neurons and *M* hidden neurons, represented by 6-*M*-4. For the MLP, 10, 20, 30, 40 hidden neurons structure were implemented respectively. For the RBFN, the spread constant of the Gaussian functions was varied as detailed in the results table (in parentheses).

Using this dataset, the performance of the two classifiers was compared. The results are listed in Table 5-7.

	MLP				RBFN			
Number of hidden neurons	10	***20***	30	40	150 (0.5)	99 (1)	***49 (2)***	49 (3)
Training error	34.3	***17.3***	14.7	16	0	12	***16***	18.7
Testing error	35.3	***17.3***	18.7	19.3	18	18	***17.3***	22

Table 5-7. Classification error rate (%) for Cooling system diagnosis.

For this task, to achieve a low classification error while keeping the classifier size small, the structure of the classifiers were 20 hidden neurons for the MLP and 49 hidden neurons for the RBFN. From these results, a minimum 'testing error' of 17.3% was obtained for each classifier, that is, the minimum classification error using the (unseen) test dataset was 17.3% for both techniques. This classification error is primarily caused by the confusion between the classes 'normal' and 'thermostat stuck open' (with a separability of 0.67, see Table 5-2), providing

empirical confirmation that neither classifier can effectively classify classes which have low separability.

c) BC case study

Both of the classifiers used for the cancer study had 9 input neurons, 2 output neurons and M hidden neurons, represented by 9-M-2. For the MLP, structures with 3, 4, 8 and 15 hidden neurons respectively were implemented with learning parameters as described in Section 2.4. For the RBFN, the spread constant was set to 4.0, 5.0 and 7.5 respectively. The orthogonal least squares algorithm was used to find the appropriate number of hidden neurons for the RBFN classifier. Table 5-8 lists the classification error rates.

	MLP				RBFN		
Number of hidden neurons	3	***4***	8	15	59 (4.0)	51 (5.0)	***48 (7.5)***
Training error	2.0	***1.3***	1.3	0.3	1.7	0.7	***1.3***
Testing error	3.3	***2.7***	2.7	3.3	1.7	1.7	***1.3***

Table 5-8. Classification error rate (%) for Breast cancer.

From the results, it is seen that the RBFN classifiers achieved a slightly smaller classification error rate than MLP classifiers for the breast cancer data. However the difference in classification error rates between all classifiers is small and all classifiers can provide good classification results (below 3.3% error rate).

From Table 5-8, the 'best' classifier of each type was selected, that is, the MLP with 4 hidden nodes and the RBFN with 48 hidden nodes. Again, for this problem, the number of hidden nodes required for the RBFN is much larger than for the MLP.

5.4.3 Basic classifier performance

The basic performance of MLP and RBFN classifiers was compared in this section, using three disparate datasets. The results demonstrated that both classifiers exhibit similar classification error rates. It was also noted that MLP always requires fewer hidden nodes than RBFN.

5.5 Conclusions

By analysing the characteristics of CMFD applications and the hardware constraints of embedded systems, a comprehensive set of classifier comparison criteria were identified in this chapter. These criteria consist of classification error, dealing with unknown and multiple faults, working with limited data size, processor requirements and memory requirements. In addition, datasets for three suitable classification problems were described.

In order to obtain the classifier structures for assessing hardware requirements of MLP and RBFN classifiers in the next chapter, this chapter also conducted a set of comparative experiments to establish basic classifier performance.

This material forms the basis of the experimental studies in the next two chapters. Chapter 6 investigates the hardware requirements when MLP and RBFN classifiers are implemented on embedded systems. Chapter 7 compares the performance of MLP and RBFN classifiers with respect to the criteria characterising the CMFD applications.

6
CLASSIFIER COMPARISONS: HARDWARE CONSTRAINTS

6.1 Introduction

Embedded CMFD systems, based on microcontrollers in particular, are increasingly found in a variety of plants (Chan, *et al*, 1997; Flammini, *et al*, 2001). As discussed in Chapter 1, such systems often have limited memory and CPU power. These hardware limits pose a practical challenge to the implementation of classifier in embedded CMFD systems (Dash & Venkatasubramanian, 2000; Kobayashi, *et al*, 2002).

In measuring the processor requirement of neural networks, some previous studies have used computer time (for example, Michie *et al*, 1994; Mak *et al*, 1993). However, the use of computer time makes results difficult to compare when classifiers are implemented on different computers. In this thesis, 'flops' (floating-point operations) are used to measure the processor requirements. Because the number of flops for a classifier solely depends on its algorithm (Rathbun, *et al*, 1997), the speed index of the classifier will be independent of the particular processor used and the results could be directly applicable elsewhere.

To evaluate memory requirements, all classifiers are implemented on two common types of microcontroller and measurements of ROM and RAM memory are made from the compiler outputs.

Note that, at the end of this chapter, consideration is given to the link between CPU and memory requirements and system power consumption.

6.2 Processor requirements

This section begins with an analysis of processor requirements for MLP and RBFN classifiers on the basis of the classifier structure itself, and then goes on to discuss how to measure the processor requirements effectively. The processor requirements of the classifiers are then experimentally compared on the three case studies described in Chapter 5.

6.2.1 Analysis of processor requirements

The processor requirement of a classifier can be assessed in two phases, training and classification. The processor requirement in the training phase is determined by the particular algorithm employed (Marzi, 2002) and is difficult to formulate due to the complexity and variety of learning algorithms. Here only the processor requirement in the classification phase is theoretically considered. The training phase will be evaluated in empirical experiments.

Consider all neural classifiers with n input nodes, c output nodes and m hidden nodes. The processor time required for floating (add, multiply, divide and

exponent) operations is τ_a, τ_m, τ_d, τ_e respectively. Then the time required for the classification of an input sample can be formulated as follows.

For MLP the time is t_m:

$$\begin{aligned} t_m = &\left(\tau_m \cdot n + \tau_a \cdot (n-1) + \tau_a\right) \cdot m + \left(\tau_e + \tau_a + \tau_d\right) \cdot m \\ &+\left(\tau_m \cdot m + \tau_a \cdot (m-1) + \tau_a\right) \cdot c + \left(\tau_e + \tau_a + \tau_d\right) \cdot c \end{aligned} \tag{6-1}$$

For RBFN the time is t_r:

$$\begin{aligned} t_r = &\left(\left(\tau_a + \tau_m + 2\tau_m + \tau_d\right) \cdot n + \tau_a \cdot (n-1) + \tau_e\right) \cdot m \\ &+\left(\tau_m \cdot m + \tau_a \cdot (m-1) + \tau_a\right) \cdot c \end{aligned} \tag{6-2}$$

Assuming add, multiply and divide require approximately the same clock cycles, $\tau_a \approx \tau_m \approx \tau_d = \tau$ (Pont, 2001; Intel51; Intel96), the above formulae may be simplified as follows:

$$\begin{aligned} t_m &\approx 2(nm + mc + m + c) \cdot \tau + (m + c) \cdot \tau_e \\ &\approx 2(n + c) \cdot m \cdot \tau + (m + c) \cdot \tau_e \end{aligned} \tag{6-3}$$

$$\begin{aligned} t_r &\approx (6mn + 2mc - m) \cdot \tau + m \cdot \tau_e \\ &\approx (6n + 2c) \cdot m \cdot \tau + m \cdot \tau_e \end{aligned} \tag{6-4}$$

From the above equations, if all the classifiers have the same number of hidden nodes, and if c is small, then the MLP requires lower processor resources. However, RBFNs usually require many more hidden nodes than MLPs (as seen in the previous chapter) and may thus be expected to require more CPU resources.

To consider the processor requirements on a specific microcontroller, the classification of MM data on standard 8051 microcontroller (Calcutt, 1998) is chosen as an example. As shown in Section 5.4.2, there are 4 hidden nodes for MLP and 50 for RBFN. Then:

$$t_m = 48\tau + 7\tau_e$$
$$t_r = 900\tau + 50\tau_e$$

The calculation of τ_e is at least 20 times longer than that of τ (Microsoft, 1992), say $\tau_e = 20\tau$, then:

$$t_m = 188\tau, \quad t_r = 1900\tau,$$

The floating point operation of τ requires 200 cycles on an 8051 device, assuming the code was compiled with the Keil compiler (Pont, 2001). Then, at 12 MHz on a standard 8051, τ=0.0002 second and therefore:

$$t_m = 0.0376\text{s}, \quad t_r = 0.38\text{s},$$

This gives an approximate analysis for the processor requirements of the classifiers.

However, in the computer industry, the measure "floating-point operations" (flops) per second is often used to measure a computer's ability to perform calculations with floating point numbers (Microsoft, 1992). Since the implementation of the algorithm of a classifier consists of certain numbers of 'flops' and the number is uniquely determined by the algorithm complexity (Rathbun, *et al*, 1997), 'flops' can be used to indicate the processor requirements.

The advantage of using 'flops' is that the measure will be independent of the particular computer used. Thus it makes the processor requirements of the classifiers obtained by different people or on different computers more easily comparable. Therefore, 'flops' was used to represent the processor requirements in the following experiments.

6.2.2 Experiments

This section presents results for comparing the processor requirements of MLP and RBFN classifiers on three classification problems. For each problem, two datasets were used. The classifiers were trained using the Training dataset and tested using the Test 1 dataset as described in Section 5.4.

In the following results, 'Training flops' are the total number of flops required for training the whole network and 'Classification flops' are measured per sample.

The tables also list classification error rates to help select the proper classifier structure, that is, the proper number of hidden nodes for neural classifiers.

a) MM case study

MLPs were trained and tested on the MM datasets using these numbers of hidden neurons: 3, 4, 6, 8 and 15. The results are listed in Table 6-1.

For RBFN the maximum number of hidden neurons were set to 50 and various spread constant values were tried. The results are listed in Table 6-2.

Hidden neurons	3	4	***6***	8	15
Testing error (%)	39.7	1	***0.3***	0.3	0.3
Training flops	$3.76 \cdot 10^8$	$4.64 \cdot 10^8$	$\boldsymbol{6.43 \cdot 10^8}$	$8.21 \cdot 10^8$	$1.45 \cdot 10^9$
Classification flops	66	82	***114***	146	258

Table 6-1. Processor requirement of MLP for MM.

Spread constant	0.01	***0.025***	0.05	0.1
Testing error (%)	5	***0.33***	0.33	0.33
Training flops	$9.59 \cdot 10^7$	$\boldsymbol{9.59 \cdot 10^7}$	$9.59 \cdot 10^7$	$9.59 \cdot 10^7$
Classification flops	1059	***1059***	1059	1059

Table 6-2. Processor requirement of RBFN for MM.

To compare the processor requirements, the classifier with the minimum error rate is identified from MLP and RBFN, and listed in ***italic bold*** font in the above two tables as well as following tables.

Note that, in this case, for the best classification error rate, the training 'speed' of the MLP is 6.7 times slower than that of the RBFN, but the testing speed of the MLP is 9.3 times faster than RBFN (primarily because the RBFN has a larger number of hidden units).

b) DC case study

For MLP classifiers, the number of hidden neurons employed was: 10, 20, 30 and 40. The training flops and classification flops are listed in Table 6-3.

For RBFN classifiers, the maximum number of hidden neurons was set to 150 and the spread constant was varied. The training flops and classification flops are listed in Table 6-4.

Hidden neurons	10	***20***	30	40
Testing error (%)	35.3	***17.3***	18.7	19.3
Training flops	$7.8 \cdot 10^8$	***$1.49 \cdot 10^9$***	$2.2 \cdot 10^9$	$2.91 \cdot 10^9$
Classification flops	284	***544***	804	1064

Table 6-3. Processor requirement of MLP for DC.

Spread constant	0.5	1	***2***	3
Hidden neurons	150	99	***49***	49
Testing error (%)	18	18	***17.3***	22
Training flops	$9.81 \cdot 10^8$	$4.05 \cdot 10^8$	***$1.16 \cdot 10^8$***	$1.16 \cdot 10^8$
Classification flops	6462	4245	***2162***	2162

Table 6-4. Processor requirement of RBFN for DC.

It is observed again that, for the classifier with the best classification error rate, the training 'speed' of the MLP is 12.8 times slower than that of the RBFN, but the testing speed of the MLP is 4 times faster than that of the RBFN.

c) BC case study

For MLP classifiers, the different numbers of hidden neurons tried were: 3, 4 8 and 15. The training flops and classification flops are listed in Table 6-5.

For the RBFN classifier the maximum number of hidden neurons was set to 100 and the spread constant was varied. The training flops and classification flops are listed in Table 6-6.

Hidden neurons	3	***4***	8	15
Testing error (%)	3.3	***2.7***	2.7	3.3
Training flops	$2.50 \cdot 10^8$	$\boldsymbol{3.21 \cdot 10^8}$	$6.06 \cdot 10^8$	$5.77 \cdot 10^8$
Classification flops	96	***124***	236	432

Table 6-5. Processor requirement for MLP with BC.

Spread constant	4.0	5.0	***7.5***
Hidden neurons	59	51	***48***
Testing error (%)	1.7	1.7	***1.3***
Training flops	$1.39 \cdot 10^8$	$1.05 \cdot 10^8$	$\boldsymbol{9.51 \cdot 10^7}$
Classification flops	3246	2760	***2598***

Table 6-6. Processor requirement for RBFN with BC.

Note that, again, for the classifiers with best classification error rate, the training 'speed' of the MLP is 3.4 times slower than that of the RBFN, but the testing speed of the MLP is 21 times faster than that of the RBFN.

6.2.3 Processor requirements: discussion and conclusions

This study demonstrates that, for these three problems, the training of an MLP requires considerably more processor operations than that of a RBFN, while the testing of an MLP requires fewer processor operations. The detailed results are summarised in Table 6-7 for training and in Table 6-8 for classification.

Problem	MM	DC	BC
MLP	$6.43 \cdot 10^8$	$1.49 \cdot 10^9$	$3.21 \cdot 10^8$
RBFN	$9.59 \cdot 10^7$	$1.16 \cdot 10^8$	$9.51 \cdot 10^7$
MLP/RBFN	6.7	12.8	3.4

Table 6-7. A summary of the processor requirements for training on the case studies.

Problem	MM	DC	BC
MLP	114	544	124
RBFN	1059	2162	2598
RBFN/MLP	9.3	4.0	21.0

Table 6-8. A summary of the processor requirements for Classification on the case studies.

The processor requirements may be a significant factor in determining the applicability of each technique in embedded systems. In the cases where off-line training is possible and rapid classification ('testing') is required, the MLP may be more appropriate. However, where on-line learning is required, the RBFN may be more appropriate.

6.3 Memory requirements

Embedded systems frequently suffer from memory constraints. Although modern microcontrollers can directly address large amounts of RAM and ROM memory and memory prices have fallen, any reductions in memory requirements can directly translate into savings in the application cost, particularly in high-volume automotive applications where product cost is of great concern (Wilmshurst, 2001; Vahid & Givargis, 2002). To evaluate memory requirements, all classifiers were implemented on two common types of microcontroller and measurements of ROM and RAM memory were made from the compiler outputs.

This section begins with an analysis of the memory requirements based on the classifier structure and size of the parameter set. Then memory requirements are evaluated on both 8-bit and 16-bit microcontrollers.

6.3.1 Analysis of memory requirements

For a trained network, if the number of hidden neurons of MLP equals the number of radial basis functions of RBFN, then the two networks can be expected to have approximately the same memory requirements. For example, for *n-M-c* structure of MLP and RBFN, both require memory to store $(n+1)\cdot M+(M+1)\cdot c$ floating point network parameters.

However one should note that, for a given fault classification problem, MLPs require fewer hidden neurons than RBFNs. Indeed, a RBFN may require more

memory than an equivalent MLP because of the large number of hidden neurons and the complexity of the training algorithm.

These issues are explored in the empirical studies below.

6.3.2 Experiments

On-line training of the classifiers studied in this thesis is rarely practical, because of the long training times that can result (Marzi, 2002). In this study, the concern is only with the classification ('testing') phase of each technique. It is assumed that training is carried out off-line (perhaps on a desktop computer) and that weights have been transferred to the embedded environment. This is a common way of using such classifiers in embedded applications (Flammini, *et al*, 2001).

The commonly used microcontrollers for embedded systems are 8-bit and 16-bit devices (Calcutt, 1998; Pont, 2001). Thus the code was implemented for two modern microcontrollers: an 8-bit device (Infineon 80c515c) and a 16-bit device (Infineon 80c167c). Table 6-9 and Table 6-10 give the memory requirements of the two classifiers for the three classification problems. The last row in the tables shows the relative memory differences as percentage between MLP and RBFN classifiers.

Problem	*MM case* 2-*M*-3		*DC case* 6-*M*-4		*BC case* 9-*M*-2	
Classifier	MLP	RBFN	MLP	RBFN	MLP	RBFN
M	4	50	20	49	4	48
Memory Size	7892	8576	8080	8590	7864	8543
Difference (%)	8.67		6.31		8.63	

Table 6-9. Memory size required for classification: 8-bit microcontroller [10].

Problem	MM case 2-*M*-3		DC case 6-*M*-4		BC case 9-*M*-2	
Classifier	MLP	RBFN	MLP	RBFN	MLP	RBFN
M	4	50	20	49	4	48
Memory Size	6139	6646	6343	6642	6114	6601
Difference (%)	8.26		4.71		7.97	

Table 6-10. Memory size required for classification: 16-bit microcontroller.

6.3.3 Discussion

In these experiments, the RBFN classifier required between approximately 6.31% and 8.67% more memory than the MLP equivalent (using an 8-bit

[10] In this table and the following table, the structure of a classifier is represented as *n*-*M*-*c*, *n* is the dimension of input vector, *c* is the number of classes, *M* is the number of hidden neurons.

microcontroller) and up to approximately 8.26% more memory (using a 16-bit microcontroller).

However it should be pointed out that, if the number of hidden neurons is small (as in this experiment), a comparatively large amount of memory is required for calculation of the exponent (which takes about 3 kbytes of memory on an 8-bit microcontroller and about 3.6 kbytes of memory on a 16-bit microcontroller). If these figures are removed from the comparison (giving a result more representative of that expected for a large network), the RBFN classifiers may require even more memory than the MLP equivalent[11].

6.4 Power consumption implications

One additional observation should also be made. If an embedded RBFN implementation requires around 20 times the number of CPU operations as an equivalent MLP classifier, then, as observed, it may be possible to implement the MLP classifier more cheaply.

In addition, it should also be noted that it would be possible to implement the MLP classifier on the same hardware platform, using a much lower oscillator frequency[12]. This may be very important because, in modern designs, system

---

[11] This is confirmed in Chapter 9.

[12] This is now possible, because many microcontrollers can be used over a very wide range of oscillator frequencies: from 0 to 24 MHz, or 0 to 50 MHz (in some cases). Note that the use of '0 MHz' may seem to have little value. However, in practice, the ability of the microcontroller to operate at 0 MHz improves the chances of system recovery following disruption to the oscillator source.

power consumption is linked almost linearly to oscillator frequency (Pont, 2001). Thus, the ability to reduce the oscillator frequency can be very valuable, particularly in battery-powered systems.

Similar, but less dramatic, reductions in power consumption can also be obtained through reduction in memory requirements (Vahid & Givargis, 2002).

6.5 Conclusions

In this chapter the hardware requirements of studied classifiers have been assessed in terms of processor requirements and memory requirements. On the basis of the results obtained, it is clear that in terms of memory requirements, the MLP requires less memory than RBFN. Also, the processor requirements for the MLP are considerably less than those for the RBFN. In concluding the chapter, the link between processor requirements (and, to a less extent memory requirements) and system power consumption was also considered.

The next chapter shifts the comparison focus to the characteristics of CMFD applications.

7
CLASSIFIER COMPARISONS: CMFD CHARACTERISTICS

7.1 Introduction

The experiments described in this chapter are concerned with the following CMFD characteristics:

1) The ability to detect unknown faults.
2) The ability to deal with multiple faults.
3) The effects of dataset size on generalisation ability.

7.2 Working with 'unknown' faults

In the previous classification experiments (Section 5.4), the problems were studied with the implicit assumption that all classes in the system are exhaustively known *a priori* and that only a single fault may occur at a time. However as discussed in Chapter 1, one significant difference between generic classifier tasks and CMFD applications is that, in the latter case, it is often difficult to obtain information about all possible system faults *a priori*. As a result many practical CMFD systems must respond 'appropriately' in situations where faults not evident in the training set are present (Dash S & Venkatasubramanian, 2000; Tarassenko, *et al*, 2000). In this section two experiments were conducted to

investigate the performance of MLP and RBFN classifiers in the presence of such unknown faults. The consideration of unknown faults begins with an analysis of the decision boundary property of a classifier. Based on this property, the capability of a classifier to deal with unknown classes is predicted.

7.2.1 Geometrical analysis of decision boundary forming

Before these networks are used for fault classification, it is important to be aware of the difference in their inherent decision making properties. The MLP partitions the input space into decision region using hyperplanes while RBFNs use hyperspheres (Leonard & Kramer, 1990; Looney, 1997; Bishop, 1995). This fundamental difference has little effect on interpolation, either a well-trained MLP or RBFN can be expected to perform well when classifying novel (unseen, non-training) samples which fall in the range of the training dataset. This was observed in the basic performance comparison in Section 5.4. However, the underlying differences in these classifiers can be expected to have a greater impact on extrapolation performance. That is, when classifying samples outside the range seen in the training dataset. Such samples are inevitable in practical CMFD applications.

To illustrate the underlying differences in the operation of the two classifiers, consider a two-dimensional measurement space. As shown in Figure 7-1, within the measurement space, the distribution of samples of normal condition (region N) and fault conditions (region A and B for fault 1, region C and D for fault 2) is assumed to be known.

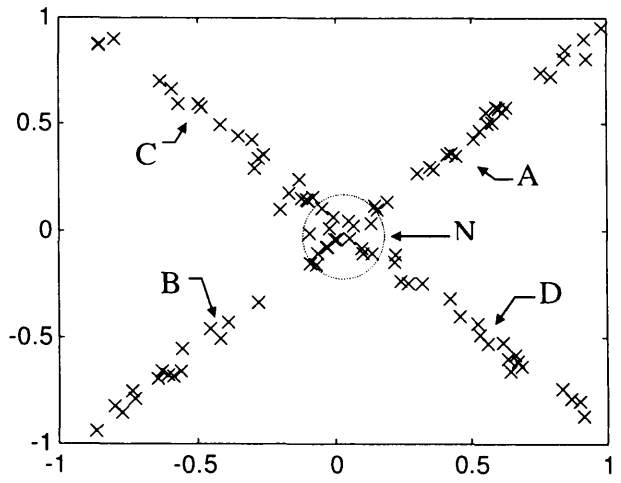


Figure 7-1. Distribution of three classes in two-dimension space.

The networks for this fault classification problem have 2 input neurons and three output neurons and are assumed to be trained using available data. After training, MLPs partition the input space with hyperplanes and so the decision boundaries are unbounded (the decision surfaces for each MLP class are shown in Figure 7-2). This can be seen in Figure 7-2(b), for example, where the classifier will produce high values for samples not only in the region of the training samples but also for samples some distance away ('some distance' may be infinitely far in some directions). On the other hand, since RBFNs partition the input space using hyperspheres, their decision boundaries are bounded (the decision surfaces for each RBFN class are shown in Figure 7-3). From the figure, it is seen that RBFN classifier produces high values for samples only from the regions covered by training samples. Figure 7-3 (d) clearly shows that the decision boundaries are closed.

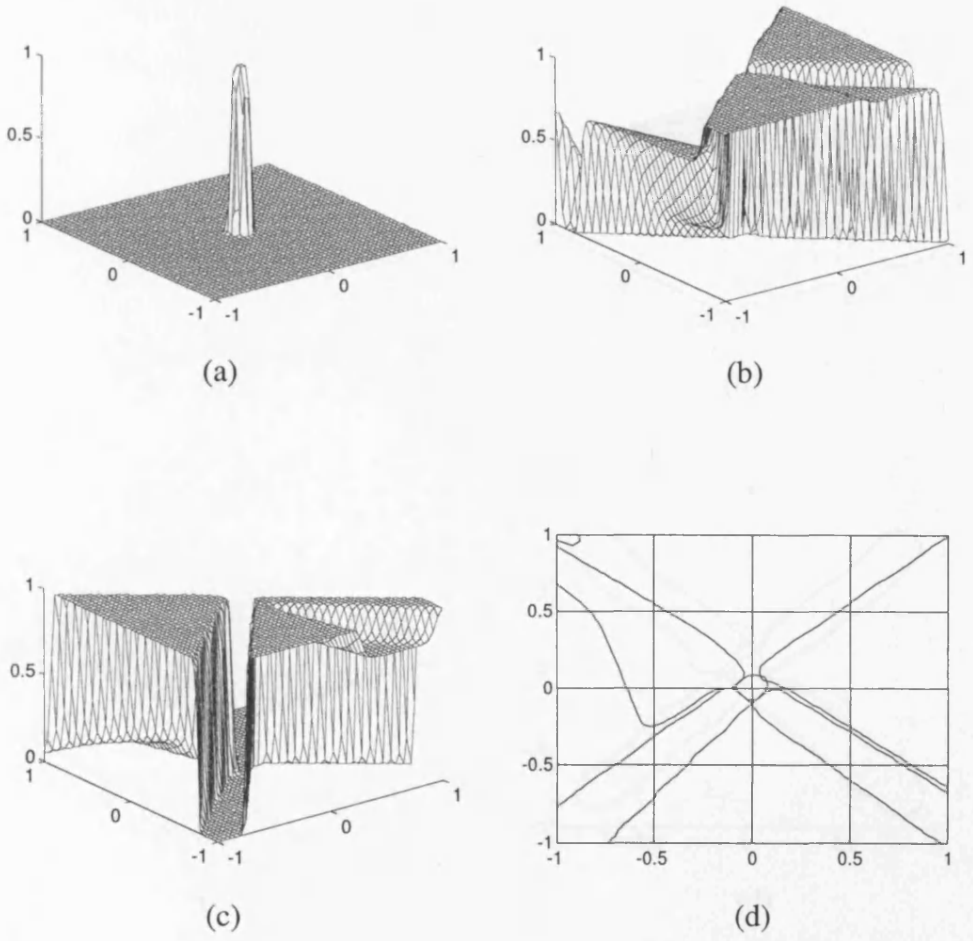


Figure 7-2. Decision surface of MLP.
(a) Decision surface for 'Normal' condition,
(b) Decision surface for 'Fault 1',
(c) Decision surface for 'Fault 2',
(d) Contour of Decision surfaces.

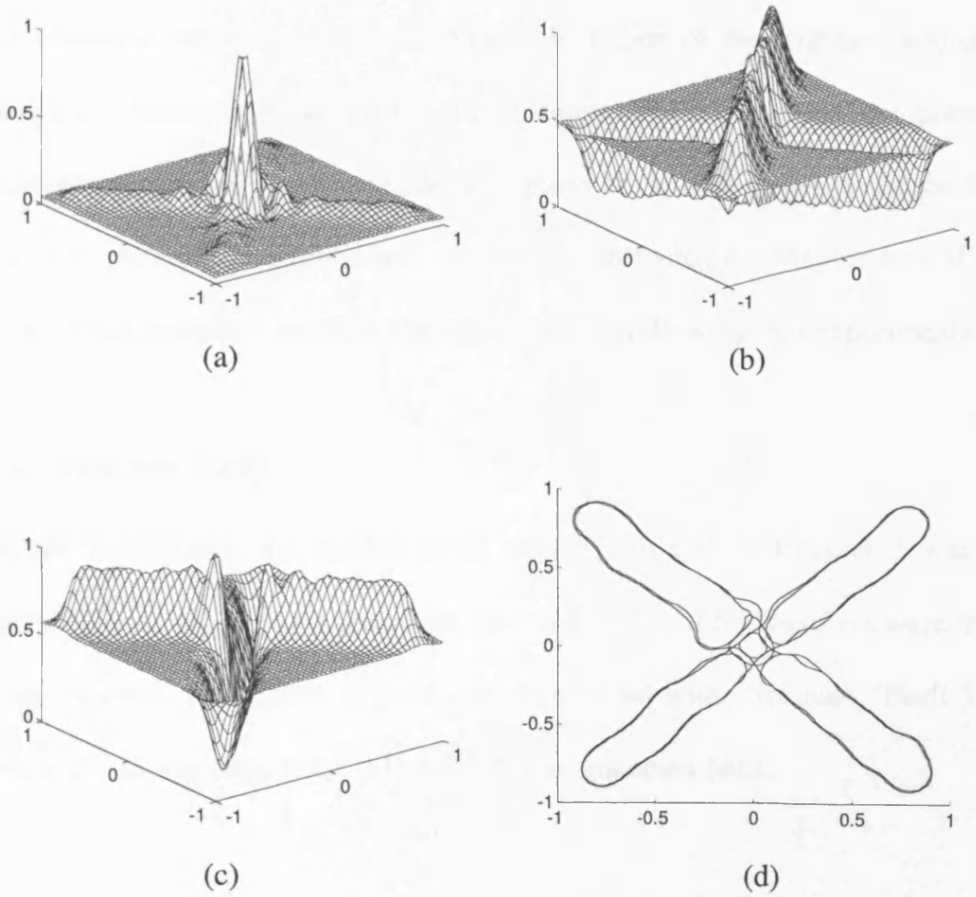


Figure 7-3. Decision surface of RBFN.
(a) Decision surface for 'Normal' condition,
(b) Decision surface for 'Fault 1',
(c) Decision surface for 'Fault 2',
(d) Contour of Decision surfaces.

A further point of note is that the decision surface of the MLP is 'uncontrolled', that is, for the MLP different training may result in different decision surfaces.

After training the networks may be used for fault classification. It is clear from Figure 7-2 and Figure 7-3, that both MLP and RBFN could provide good classification results for novel samples in the region of the original training data samples. However if a novel 'out of range' fault occurs in the plant, the unexpected samples will lie outside the regions N, A, B, C or D. While the RBFN may still provide an 'appropriate' result for such faults, the MLP is less likely to do so. This analytical result is assessed using the following two experiments.

7.2.2 MM case study

In this experiment, the mathematical model presented in Chapter 5 was used again. However, in this experiment the MLP and RBFN classifiers were trained with 'Normal' and 'Fault 1' only, and then tested with 'Normal', 'Fault 1' and 'Fault 2'. In this case, Fault 2 is treated as an unknown fault.

Both the classifiers have two inputs and two outputs. The number of hidden neurons is 4 for the MLP and 19 for the RBFN (with spread constant 0.09).

The classification results are presented in the form of a confusion matrix (Blayo, *et al*, 1995). The confusion matrix **C** is:

$$C_{ij} = \varepsilon(\text{class} = j \mid \text{class} = i)$$

where ε is the normalised number of the classification result *j*, if the class *i* is given.

Table 7-1 and Table 7-2 list the confusion matrix of classification for the test data.

	Normal	Fault 1	Fault 2 (unknown)
Normal	1.0	0	0
Fault 1	0	1.0	0
Fault 2 (unknown)	0.82	0.13	0.05

Table 7-1. Confusion matrix (MLP).

	Normal	Fault 1	Fault 2 (unknown)
Normal	1.0	0	0
Fault 1	0	1.0	0
Fault 2 (unknown)	0.20	0	0.80

Table 7-2. Confusion matrix (RBFN).

From these tables[13] it is notable that the MLP classified 82% of the 'Fault 2 (unknown fault)' samples as 'Normal', and that only 5% of these faults were classified correctly. By contrast, the RBFN classified only 20% of the 'Fault 2'

[13] The absolute figures for classification accuracy would be different if the distribution of the data was different. However this does not influence the conclusion of the classifiers' ability when dealing with unknown faults, because the ability of a classifier to deal with such unknown faults is determined by the classifier's inherent decision behaviour.

class as 'Normal', and correctly classified 80% of these as unknown faults. The overall performance of the RBFN classifier for this study was 93% while that of the MLP was approximately 68%. This experiment supports the theoretical prediction that RBFN can provide a good indication for 'unknown faults', while MLP is unlikely to do so.

7.2.3 DC case study

In this experiment, the nonlinear model of the diesel cooling system presented in Chapter 5 is employed again. However, both the classifiers were trained with 'Normal' and 'Fan off' data only, and then tested with 'Normal', 'Fan off', 'Thermostat stuck open' and 'Pump fault'. Thus, in this study, 'Thermostat stuck open' and 'Pump fault' examples are treated as unknown faults.

Both classifiers had six input nodes and two output nodes. The number of hidden neurons was 20 for MLP and 50 for RBFN with spread constant 0.5. Table 7-3 and Table 7-4 list the confusion matrices for test data.

	Normal	Fan off	Thermostat stuck open (unknown)	Pump fault (unknown)
Normal	1.0	0	0	0
Fan off	0	1.0	0	0
Thermostat stuck open (unknown)	1.0	0	0	0
Pump fault (unknown)	1.0	0	0	0

Table 7-3. Confusion matrix (MLP).

	Normal	Fan off	Thermostat stuck open (unknown)	Pump fault (unknown)
Normal	1.0	0	0	0
Fan off	0	1.0	0	0
Thermostat stuck open (unknown)	1.0	0	0	0
Pump fault (unknown)	0	0	0	1.0

Table 7-4. Confusion matrix (RBFN).

In this study there were two unknown faults, 'Thermostat stuck open' and 'Pump fault'. From these tables, it is apparent that the MLP classified 100% of the unknown faults as 'Normal'. By contrast, while the RBFN classifier also classified 'Thermostat stuck open' data as 'Normal', it correctly classified all of the 'Pump fault' as unknown faults. This is because the "Thermostat stuck open"

is confused with "Normal" (with separability value of 0.67, see Table 5-2), while the "Pump fault" is well separated from the known classes of "Normal" and "Fan off" (both with large separability, see Table 5-2). The overall performance of the RBFN classifier was 75% while that of the MLP was around 50% for this study. The results demonstrate that RBFN classifier can detect unknown faults, if they are well separated from known classes.

7.2.4 Unknown faults: conclusions

Where unknown faults are concerned, these studies confirm the theoretical prediction that RBFNs can provide accurate classification results for unknown faults if they are well separated from known classes, but still (inevitably) performs poorly for those unknown faults which overlap with known classes. On the other hand, MLPs always attempt to classify samples of unknown faults into known classes since the decision boundaries of the known classes are unbounded.

7.3 Working with multiple faults

The previous experiments have considered only a single fault. It is often the case that more than one fault will occur simultaneously in a practical system (Chung, *et al*, 1994; Watanabe, *et al*, 1994; Hsu, *et al*, 1995; Maki & Loparo, 1997; Cheon, *et al*, 1993). This poses a challenge to CMFD systems because multiple faults may interfere with one another and are, as a result, more difficult to classify (Hsu, *et al*, 1995).

In this section the performance of MLPs and RBFNs for dealing with multiple faults is compared. The comparison was carried out on the diesel engine cooling system model with which simultaneous radiator and pump faults were simulated.

7.3.1 The experimental dataset

To compare the performance of neural classifiers dealing with multiple faults, the non-linear model (Chapter 5) of an engine cooling system was employed. Here the engine cooling system was assumed to experience four different conditions: normal, radiator fault, pump fault, and simultaneous radiator and pump fault.

Again six measurements (as described in Chapter 5) were used for classification. Both training and testing datasets were generated from the model, and each contained 400 samples. Each of the four conditions was represented by 100 samples. Table 7-5 lists the separability of classes on the whole dataset.

	Normal	Radiator	Pump	Radiator & Pump
Normal	1	1.68	137.48	95.35
Radiator	1.68	1	129.04	87.54
Pump	137.48	129.04	1	37.22
Radiator & Pump	95.35	87.54	37.22	1

Table 7-5. Separability Matrix of Multiple Faults.

Table 7-5 shows that there is low separability (with value of 1.68) between Normal and Radiator fault, but the separability is high between other classes. This explores how neural classifiers perform for classes with high separability when multiple faults exist.

7.3.2 The classifier structure

When considering only a single fault, the input vector is classified as being of the corresponding class with the highest value for the classifier outputs. For multiple classes, the output of a classifier must be interpreted in a different way:

1. An output node represents a single class, where 1 indicates the occurrence of the class and 0 non-occurrence.
2. A threshold, τ, is introduced. If the output of a node exceeds the threshold τ, the output value is rounded to 1, otherwise it is rounded to 0.

For the above cooling system problem, the networks (again) have the same input measurements and have three output neurons to represent normal, radiator fault and pump fault. Multiple faults are represented by the combination of output neurons, for example, for three classes, {0 1 0} is used for class 2 and {0 0 1} for class 3, so in the case of multiple faults of class 2 and 3, the values of output neurons are expected to be {0 1 1}. In this way multiple faults are classified by MLPs and RBFNs. Table 7-6 lists the output patterns for cooling system fault diagnosis.

Class	Output pattern
normal	1 0 0
radiator	0 1 0
pump	0 0 1
radiator & pump	0 1 1

Table 7-6. Representation of multiple classes.

The number of hidden neurons of MLP and the width of radial basis function are determined by trial and error. All other network parameters have the values described in Section 2.4.

7.3.3 Results

The trained networks were evaluated using the testing dataset. In assigning a sample to a class, an output threshold of 0.5 (0.5 is intuitive for output value between 0 and 1, Haykin 1994) was used for all the neural classifiers. A particular output is said to represent a particular class if one output neuron value exceeds this threshold. If a sample cannot be assigned to any of the pre-defined classes (normal, radiator fault, pump fault, radiator & pump faults), it is treated as an instance of an 'unknown fault' condition.

Table 7-7 and Table 7-8 list the confusion matrices for the classifications by MLP and RBFN respectively.

	normal	radiator	pump	Radiator & pump	unknown
normal	0.51	0.4	0	0	0.09
radiator	0.17	0.73	0	0	0.1
pump	0	0	1.0	0	0
Radiator & pump	0	0	0	1.0	0

Table 7-7. Confusion matrix of MLP classification.

	normal	radiator	pump	Radiator & pump	unknown
normal	0.44	0.51	0	0	0.05
radiator	0.22	0.77	0	0	0.01
pump	0	0	1.0	0	0
Radiator & pump	0	0	0	1.0	0

Table 7-8. Confusion matrix of RBFN classification.

From the Table 7-7 and Table 7-8, it is observed that both classifiers exhibit a similar level of performance for this multiple fault problem. The overall classification accuracy was 81% for the MLP and 80.25% for the RBFN. In fact, since the 'radiator & pump' fault has very high separability from other classes (see Table 7-5), both the classifiers provide 100 percent classification rate for this example. Because the separability between classes of Normal condition and Radiator fault is very low (see Table 7-5) there is a strong overlap between these two classes and both the classifiers performed poorly in this situation. This

experiment demonstrates that, on this problem, neural classifiers perform well regardless of single fault or multiple faults, if there is high separability between the classes.

7.4 Working with limited training data

The ability of a trained classifier to generalise correctly is known to be influenced by three key factors: the physical complexity of the problem at hand, the architecture of the network, and the size and quality of the training dataset (Haykin, 1999). Clearly the physical complexity of the problem cannot be directly controlled by designers[14]. The influence of the network architecture on the generalisation ability was studied in Section 5.4, by fixing the training data size and altering the classifier structure (with a different number of hidden nodes for MLP or different spread constant for RBFN, etc.).

This section will therefore be devoted to an investigation into the effect of training data size on generalisation. In doing this, the main focus is on determination of the size of training dataset needed to achieve good generalisation.

Intensive theoretical investigations in the effects of dataset size have resulted in a number of 'rules of thumb' that may be used to suggest how many training samples are required for successful learning. Some key theoretical results are discussed below. However, in practical CMFD applications, it is rarely possible

[14] Note, however, that the designers may influence the problem complexity through the use of appropriate pre-processing techniques as discussed in Chapters 3 and 4.

to obtain the number of samples suggested by theoretical formulae[15]. As a result empirical studies, such as that discussed in this chapter, are an important adjunct to theoretical work in this area.

7.4.1 Theoretical analysis

It is well understood that the number of samples in the training data can affect how well a network may be trained. Investigations into the impact of sample size have focused on three main areas: statistical analysis, geometrical analysis and worst-case analysis based on Vapnik-Chervonenkis (VC) theory (Smolensky, *et al*, 1996). This section will present some results about the effect of dataset size from these frameworks.

a) Geometrical view

In the process of classification, a neural network partitions the input space into regions where each region is formed by a hyperplane (or hypersphere) segment. Samples in a region belong to the same class, and each class may consist of a number of regions. Thus, the process of training a neural network can be viewed as a process of constructing optimal hyperplanes.

Because a hyperplane separates classes from each other, samples near a class boundary, called boundary samples, are important for identifying hyperplanes (Lee & Landgrebe, 1997). Based on this geometrical view, we can determine the

---

[15] As discussed in Chapter 1, data for fault classes may be particularly difficult to obtain, not least because it may involve permanent damage to expensive equipment.

minimum number of samples required for successful training by determining how many boundary samples are needed to identify the hyperplanes.

Given the input dimensionality n, the number of hidden nodes m in a neural network and the number of clusters M_c of input samples (generally $M_c \geq$ the number of classes or output nodes), Mehrotra *et al* (1991) determined that the number of boundary samples N_B required for successful classification is proportional to:

$$N_B = \Omega\left(\min(m,n) \cdot M_c\right) \tag{7-1}$$

This method emphasizes boundary samples, however in general most samples are not boundary samples. The nature of the distribution of samples within clusters determines the proportion of the number of boundary samples, hence the overall number of input samples required is likely to be more than $\min(m,n) \cdot M_c$.

In practice, since the actual distribution of input samples is unlikely to be known, it is difficult to determine the overall number of input samples needed to generate the required boundary samples. One solution may be to pre-process the overall input samples to identify the boundary samples (Hara & Nakayama, 1998) then use these samples to train the network[16].

[16] One potential benefit of this approach is that it will reduce the training time due to a small number of training samples being used.

b) VC dimension view

The Vapnik-Chervonenkis dimension is a measure of the capacity of the family of classification functions realised by the learning machine (Haykin, 1999). Stated in terms more specific to neural classifiers, the VC dimension of the set of classification functions is the maximum number of training samples that can be learned by the neural classifier without error, for all possible binary labelling of the classification functions.

Using the measure of VC dimension, Baum & Haussler (1989) presented a formula to find the upper bound of training samples required for reasonable generalisation. Assuming that the neural classifier has a total of M nodes in the hidden and output layer, and a total of W weights, they showed that, if some number N_{VC} of samples given by:

$$N_{VC} \geq \frac{W}{\varepsilon} \log_2 \frac{M}{\varepsilon} \tag{7-2}$$

can be learned by the network such that a fraction $1-\frac{\varepsilon}{2}$ are correctly classified (where $0 < \varepsilon \leq \frac{1}{8}$) then there is a high probability that the network will correctly classify a fraction $1-\varepsilon$ of future samples drawn from the same distribution of training samples.

Instead of using the VC dimension, Takahashi & Gu (1998) introduced the Boolean interpolation (IP) dimension d_{IP}. IP dimension is the supremum of the smallest number of input samples needed for fixing the decision boundary of a

neural network. If the number of changeable parameters (weights) in a network is W, then $d_{IP} \leq W$ is true. Given an error rate $\varepsilon > 0$ and a confidence 1-δ, then the minimum samples size achieving successful learning is:

$$N_{IP} \geq \frac{1.2}{\varepsilon}\sqrt{d_{IP}\ln\frac{1}{\delta e}} + \frac{d_{IP}}{\varepsilon} \tag{7-3}$$

where ln denotes the natural logarithm, e is the base of natural system of logarithms.

c) Example

The previous sub-sections presented some of the main theoretical results in finding training samples for successful learning. This section will apply those formulae to the static fault diagnosis problem described in Chapter 5.

For this classification problem, the neural classifier has 2 input nodes, 4 hidden nodes and 3 output nodes, so the total nodes M=7 (excluding input nodes), the total weights W=27 and the number of clusters of input sample M_c=5. Assume d_{IP}=W=27, $\varepsilon = 0.05$, $\delta = 0.05$, then the number of training samples for successful learning derived from different methods is shown in Table 7-9.

Method	N_B	N_{VC}	N_{IP}
Training Samples	10	3850	716

Table 7-9. Training samples derived from different methods.

Looking at Table 7-9, Mehrotra's geometrical method says 10 boundary samples are needed for successful learning, while Baum's method shows 3850 samples are needed for the same problem.

There are no simple techniques available allowing us to identify boundary samples from a dataset. It is intuitively obvious that the number of boundary samples in a dataset may vary with the complexity (for example, shape of class boundary, size of class region, dimensionality) of the classification problem. It is probably true that more samples are required to warrant the number of boundary samples derived from Mehrotra's method, but the difference between the different techniques is still significant.

Overall, it is not clear how useful these theoretical studies are in practice. In some neural network paradigms (for example, multilayer neural networks), the VC dimension is hard to calculate (Takahashi & Gu, 1998). More generally, the theoretical bounds may be regarded more as an attempt to come to a theoretical understanding of the true behaviour of the training process, rather than as a tool for direct application in practical systems (Haussler, *et al*, 1997).

7.4.2 Experimental comparison

As discussed above, developers of practical CMFD applications still need to rely, at least in part, on empirically derived sample size bounds (Haussler, 1992; Zhang, 2000). In fact, previous experiments indicate that satisfactory generalisation error is sometimes obtained for sample sizes considerably smaller

than theoretically estimated bounds (Haussler, *et al*, 1997; Kramer & Leonard, 1990; Rao, 1999).

In the remainder of this chapter an empirical study will be described, which compared the effects of training sample size on the generalisation error of MLP and RBFN classifiers in CMFD applications. The aim of the experiments is to investigate how the number of samples affects the performance, this in turn determines the selection of an appropriate classifier for the available data size. The experiments are made on static fault diagnosis data and cooling system fault diagnosis data.

a) MM case study

The structures of the classifiers were determined based on the results in Section 5.4. Specifically, the number of hidden nodes was 4 for the MLP, and the spread constant was 0.025 for the RBFN. All other parameters use default values.

Again, training samples of 30, 100, 300, 600 were generated from Equation (5-15). After training, the two testing datasets described in Section 5.4 were used for exploring the performance of the classifiers. Table 7-10 summarises the experimental results.

	MLP				RBFN			
Samples	30	100	300	600	30	100	300	600
Training error	0	0	0	0.33	0	0	0.33	0
Error for Test 1	6.0	2.33	1.0	0.67	28.67	10.33	0.33	1.0
Error for Test 2	22.33	28.67	17.67	32.33	30.33	27.67	1.67	1.0

Table 7-10. Performance vs. Samples on MM data.

From Table 7-10, it can be seen that the RBFN requires more training samples to achieve good classification performance, and that the number of samples strongly affects the generalisation ability of RBFN, but less so for MLP. The extrapolation ability of MLP cannot be increased by increasing the number of training samples.

b) DC case study

The structures of classifiers were determined based on the results in Section 5.4. Specifically, the number of hidden nodes was 20 for MLP and spread constant was 2 for RBFN. All other parameters use default values.

Again training samples of 32, 100, 300, 600 were generated from engine cooling system model described in Chapter 5. After training, the testing dataset used in Section 5.4 was used for examining the performance of the classifiers. Table 7-11 summarises the experimental results.

	MLP				RBFN			
Samples	32	100	300	600	32	100	300	600
Training error	25	0	17.3	9.17	3.1	0	16	10.17
Test error	28.33	22.5	17.3	20	49.67	32.67	17.3	19.7

Table 7-11. Performance vs. *Samples on DC data.*

From Table 7-11, it can be observed that the RBFN generalised poorly if the training data size was small. The number of samples strongly affects the generalisation ability of the RBFN, but less so for the MLP.

7.4.3 Discussion

Suppose the input dimension is n, the number of classes is c, the total number of training samples is N and the training samples per class is N_c. To analyse these results, the size of the training dataset is described using the ratio of the training samples per class to the input dimensions, N_c/n, which is generally accepted as an indicator of characterising data size in the practice of classifier design (Jain, 2000).

The corresponding values of N_c/n for the experiments in the last section are listed in Table 7-12 and Table 7-13.

N	30	100	300	600
N_c	10	33	100	200
N_c/n	5	16.5	50	100

Table 7-12. Size of training data for MM.

N	32	100	300	600
N_c	8	25	75	150
N_c/n	1.33	4.1	12.5	25

Table 7-13. Size of training data for DC.

The size of a dataset is often described using linguistic terms such as 'small' and 'large'. Since it is difficult to say exactly how many samples should be in, for example, a 'small' training set, a fuzzy set (Zadeh, 1973; Ross, 1995) is introduced for this description. Here three fuzzy sets are assigned to N_c/n. The corresponding membership functions μ for N_c/n is defined as in Figure 7-4. The supports and boundaries[17] of the membership functions for the fuzzy sets are intuitively determined based on the experiments.

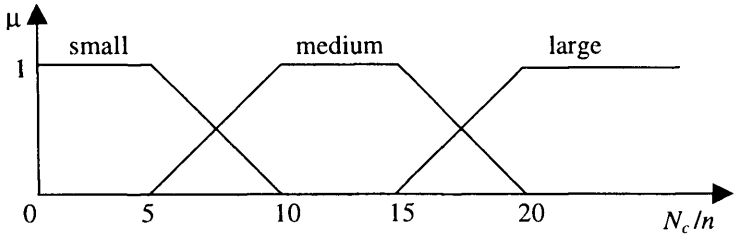


Figure 7-4. Size of training data characterised by fuzzy sets.

Referring to MM data in Table 7-12, it can be said that the size of training data is small for 30 samples, medium for 100 samples and large for 300 and 600 samples.

---

[17] The support of membership function for a fuzzy set A comprises those elements x of the universe such that $\mu_A(x) > 0$. The boundaries of membership function for a fuzzy set A comprise those elements x of the universe such that $0 < \mu_A(x) < 1$.

While for DC data in Table 7-13, it can be said that the size of training data is small for 32 and 100 samples, medium for 300 hundred samples and large for 600 samples.

Examining Table 7-10 and Table 7-11 and the values of N_c/n, conclusions about the impact of training set size to the classification performance can be made as follows:

1) For a small-sized training dataset, MLP clearly outperforms RBFN;
2) For a medium-sized training dataset, MLP slightly outperforms RBFN.
3) For a large-sized training dataset, both classifiers provide comparable classification performance.

Taking the above observations into account, it is appropriate to use an MLP for the cases with small- and medium-sized set of training data, and a RBFN for cases with large- and medium-sized set of training data.

7.4.4 Data size: conclusions

Neural classifiers are data-driven techniques. The size of the dataset strongly affects the generalisation ability of a classifier.

In this section, some theoretical bounds for training data size were discussed. These theoretical bounds tend to suggest that large amounts of training data are required. Because of practical limitations in the availability of training data, the

requisite number of samples is usually difficult to achieve in real CMFD problems.

This section also empirically compared the influence of dataset size to the generalisation ability of the studied classifiers. Based on the results, it may be concluded that it is more appropriate to use the RBFN for the cases of medium and large number of training samples, and the MLP for cases of small and medium number of training samples.

7.5 Conclusions

In this chapter, the results from a series of empirical studies intended to consider the suitability of MLP and RBFN classifiers for use in CMFD applications have been presented. The empirical studies considered the ability to deal with unknown or multiple faults, and the effects of training dataset size.

Overall, on the basis of the results obtained in these studies, it can be seen that each form of classifier has both strengths and weaknesses, and that neither is suitable for use in all CMFD applications.

So far we have addressed issues pertinent to the first and the second stages of designing embedded CMFD systems. The next chapter will move on to the final stage, the post-processing of classifier outputs.

8
SELECTING THRESHOLDS FOR RBFN CLASSIFIERS

8.1 Introduction

The results obtained in Chapter 7 verified the theoretical predication that the RBFN classifiers have the ability to identify classes which had not been seen in training data. To achieve this, the outputs of the classifier were interpreted by applying a threshold to the output vector. If the value of an output neuron exceeds the given threshold, then an example of the corresponding class is said to have occurred (Joshi, *et al*, 1997; Cheon, *et al*, 1993; Maki & Loparo, 1997; Isermann, 1997).

In addition to the ability to represent unknown classes, this threshold-based classification scheme has another important application in multiple fault classification: this was also discussed in Chapter 7.

Despite these potential advantages, use of a threshold classifier for CMFD applications can be problematic, because the overall performance of the system depends on the use of an appropriate threshold value. In most published work in this area, thresholds are simply empirically set, usually at values of '0.5'

(Watanabe, *et al*, 1994; Haykin, 1999), which does not necessarily result in optimal classifier performance (Joshi, *et al*, 1997; Theodoridis & Koutroumbas, 1999).

To improve the performance of threshold based classifiers in CMFD and other application areas, methods for identifying the optimal threshold values are required. This chapter addresses this post-processing problem by developing a method which improves the performance of RBFN classifiers in CMFD applications where an 'unknown' fault may occur. This novel technique is based on an analysis of the relationship between the behaviour of a well-trained RBFN classifier and its response to the dataset.

The chapter is organised as follows: in Section 8.2, a method for determining a suitable threshold is derived, based on theoretical considerations; the results of two empirical tests are presented in Section 8.3; the results are discussed in Section 8.4.

8.2 Theoretical considerations

A technique for reliable threshold selection for RBFN classifiers will be derived in this section. The problem of threshold selection for neural network classifiers is first formulated in Section 8.2.1. In Section 8.2.2 the decision behaviour of RBFN classifiers is mathematically and geometrically analysed. The reliable threshold selection method for RBFN classifiers is then proposed in Section 8.2.3.

8.2.1 The general problem

A basic learning problem can be represented by six components (Smolensky, *et al*, 1996): X, Y, A, $\mathcal{H}$, $\mathcal{P}$, and L. The first four components are the instance (input vector), outcome, decision, and decision rule, respectively. X is an arbitrary set, Y and $A \in \{0,1\}$, and $\mathcal{H}$ is a family of functions from X into A. The fifth component, $\mathcal{P}$, is a family of joint probability distributions of $Z = X \times Y$. These represent the possible states that might be governing the generation of examples. The last component, the loss function L, is a mapping from $Y \times A$ into the real number set R.

This chapter mainly concerns the characteristics of A and L with respect to the threshold. Assume that the classifier has been well trained with samples $(z_1, \cdots, z_N)$, where $z_i = (\mathbf{x}_i, \mathbf{y}_i) \in Z$, drawn independently at random according to some probability distribution $P \in \mathcal{P}$. After training the classifier is applied to a set of samples whose class is known. Suppose the outcome of an output neuron is $\hat{y} \in [0,1]$, a threshold is applied to $\hat{y}$, then we have the hypothesis $h \in \mathcal{H}$ that specifies the appropriate action $a \in A$ as:

$$h : X \rightarrow A, \quad a(\mathbf{x}, \tau) = \theta(\hat{y} - \tau) \tag{8-1}$$

where: $\theta(\delta) = \begin{cases} 0 & \text{if} \quad \delta \leq 0 \\ 1 & \text{if} \quad \delta > 0 \end{cases}$

This definition of θ gives (8-1) the following values:

$$a(\mathbf{x},\tau) = \theta(\hat{y} - \tau) = \begin{cases} 0 & \text{if} \quad \hat{y} \le \tau \\ 1 & \text{if} \quad \hat{y} > \tau \end{cases} \tag{8-2}$$

where τ is the threshold. a equals one for indicating the class occurred or zero otherwise. Further, we consider the following loss function:

$$L(z,\tau) = L(\mathbf{y},\mathbf{a}(\mathbf{x},\tau)) = \begin{cases} 0 & \text{if} \quad \mathbf{a} = \mathbf{y} \\ 1 & \text{if} \quad \mathbf{a} \neq \mathbf{y} \end{cases} \tag{8-3}$$

and the risk function (Scholkopf, *et al*, 1997):

$$I(\tau) = \int L(z,\tau) dP(z) \tag{8-4}$$

Since the probability distribution function $P(z)$ is unknown, but random and independent sample of pairs $z_i = (\mathbf{x}_i, \mathbf{y}_i)$ are given, then from (8-4) we have instead the empirical risk function:

$$I(\tau) = \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{y}_i, \mathbf{a}_i) \tag{8-5}$$

Thus the expected risk of the decision rule (or hypothesis) is simply the probability that it predicts incorrectly, the usual notation of the error of the hypothesis.

For the above classification problem, the risk function I is obviously bounded and non-negative. Our goal is then to minimise the risk function, I, by determining the optimal value of the threshold, τ (see Section 8.2.3)

8.2.2 Behaviour of the RBFN classifier

Referring to Chapter 2 (Section 2.3), the output of an output neuron of RBFN with m hidden neurons has the form (Broomhead & Lowe, 1988; Moody & Darken, 1988):

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b \tag{8-6}$$

where ϕ are basis functions which can be one of several types (Sanchez, 1996). The weight coefficients $\mathbf{w}$ combine the basis functions into an output value, and b is a bias term.

The mostly commonly used radial basis function is the Gaussian basis function:

$$\begin{aligned} y(\mathbf{x}) &= \sum_{i=1}^{m} w_i \phi_i(\mathbf{x}) + b \\ &= \sum_{i=1}^{m} w_i \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}_i\|^2}{2\sigma_i^2} \right) + b \end{aligned} \tag{8-7}$$

Thus in (8-7) ϕ_i is the ith Gaussian basis function with centre $\mathbf{c}_i$ and variance σ_i^2.

Since the radial basis functions in the hidden layer have localised response behaviour in the input space, its response approaches zero at large radii, that is:

$$\phi_i \xrightarrow{\|\mathbf{x}-\mathbf{c}_i\| \to \infty} 0$$

Considering the most common form of coding scheme for classification using neural networks (Tarassenko & Roberts, 1994), the output value y is 1 if the input

pattern **x** belongs to the class and 0 otherwise, that is, $y \in [0,1]$ on the training data[18]. As an RBFN classifier obtains its parameters through training, this gives the following statement about the interval of bias in the output layer:

For classification using RBFN, $y = \mathbf{w}^T \phi(\mathbf{x}) + b$, *if the output is set as* $y \in [0,1]$ *on training data, then a well trained RBFN classifier will satisfy:* $b \in [0,1]$.

The validity of the above statement can be proved as follows:

Since $y \in [0,1]$, and $\phi \in (0,1]$, **w** are finite numbers,

if an input vector is far from the centers of the radial basis functions, we have

$$\|\mathbf{x} - \mathbf{c}_i\| \to \infty \Rightarrow \phi_i \to 0, \text{ for all } i, i=1,\ldots m$$

then from (8-7) we have $\lim_{\phi \to 0} y = b$

assume $b \notin [0,1]$, say $b > 1$, then $y = \mathbf{w}^T \phi(\mathbf{x}) + b \to b > 1$, which contradicts the condition of $y \in [0,1]$. Thus $b \leq 1$. Similarly we can prove that $b \geq 0$.

Thus $b \in [0,1]$.

---

[18] The condition $y \in [0,1]$ applies for the **training data only**, using the coding scheme discussed above. Even when an 'optimally' trained network is used for classification, the network output may still fall outside the interval [0,1]. In this case, the classifier can still perform well when an appropriate threshold is applied to the network output.

To understand the physical meaning of outcomes and the bias in the output layer, consider a geometric interpretation.

Suppose there is a two-class problem with inputs distributed in two dimensions as in Figure 8-1 (a). An RBFN classifier with two input and two output neurons is used for this classification problem. After training, the values of the output neurons with respect to the input variable x_1 are as in Figure 8-1 (b). The corresponding output neuron has a high value if the input is within the region of the class, while the other output neuron has a low value. Outcomes of all output neurons will be asymptotic to their bias if the input is out of their class region.

It is notable that the bias value of a trained RBFN classifier may lie outside the interval [0, 1] in some implementations. However, such classifiers will perform very poorly, as demonstrated in Sections 8.3 and 8.4. In these circumstances the RBFN classifier will generally need to be re-trained by adjusting the training parameters (that is, the number and the spread constant of the radial basis functions).

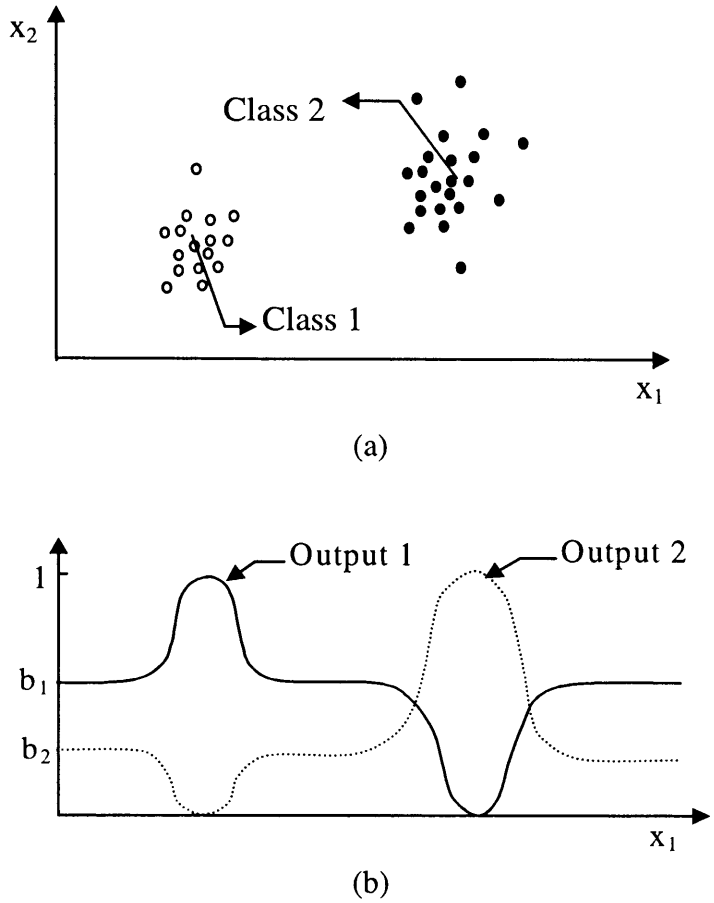


Figure 8-1. Outcomes of output neurons.

8.2.3 Reliable threshold selection for RBFN classifiers

As discussed in Section 8.2.1, for the problem of classification, our goal is to determine a threshold that tends to minimise the probability of erroneous classification in a given class. To derive the optimal threshold for a radial basis function classifier, the general model (Smolensky, *et al*, 1996) for describing the task of generalisation optimisation of a neural network is used.

The goal of designing a network is to find the network which provides the most likely explanation of the observed dataset. To do this it is necessary to try to maximise the probability:

$$P(\mathcal{N}|\mathcal{D}) = \frac{P(\mathcal{D}|\mathcal{N})P(\mathcal{N})}{P(\mathcal{D})} \tag{8-8}$$

where $\mathcal{N}$ represents the network (with all of the weights and biases specified), $\mathcal{D}$ represents the observed dataset, and $P(\mathcal{D}|\mathcal{N})$ is the probability that the network $\mathcal{N}$ would have produced the observed data $\mathcal{D}$. Applying the monotonic logarithm transformation to (8-8), we have:

$$\ln P(\mathcal{N}|\mathcal{D}) = \ln P(\mathcal{D}|\mathcal{N}) + \ln P(\mathcal{N}) - \ln P(\mathcal{D}) \tag{8-9}$$

Thus maximising (8-9) is equivalent to maximising (8-8).

Since the probability distribution of the data is not dependent on the network, $\ln P(\mathcal{D})$ will have no contribution to the maximising solution of $\ln P(\mathcal{N}|\mathcal{D})$ and is dropped from (8-9).

The second term of (8-9), $\ln P(\mathcal{N})$, is a representation of the probability of the network itself: that is, it is the *a priori* probability or *a priori* constraint on the network. Since our method assumes that the classifier has been well trained and our purpose is to determine an optimal threshold for the trained RBFN classifier, we will also drop this term.

The first term of (8-9), $\ln P(\mathcal{D}|\mathcal{N})$, represents the probability of the data given the network: that is, it is a measure of how well the classifier accounts for the data. Therefore the threshold selection problem is equivalent to a requirement to maximise $\ln P(\mathcal{D}|\mathcal{N})$.

Further, if the data are broken into two parts, the output **y** and the input **x**, then:

$$\begin{aligned} \ln P(\mathcal{D}|\mathcal{N}) &= \ln P((\mathbf{x},\mathbf{y})|\mathcal{N}) \\ &= \ln P(\mathbf{y}|\mathbf{x}\wedge\mathcal{N}) + \ln P(\mathbf{x}) \end{aligned} \tag{8-10}$$

where '$\mathbf{x}\wedge\mathcal{N}$' stands for inputting **x** to $\mathcal{N}$.

Finally, suppose that the input **x** does not depend on the network, then the last term of (8-10) has no effect in maximising $\ln P(\mathcal{D}|\mathcal{N})$. Therefore, we need only maximise the first term $\ln P(\mathbf{y}|\mathbf{x}\wedge\mathcal{N})$.

For the classification problem, the output vectors, **a**, defined in (8-2), consist of a sequence of 0's and 1's. In this case, we imagine that each element of the classifier output, *a*, represents the probability that the corresponding element of the desired output *y* takes on the value 0 and 1. Then the probability of the output given the network, for a problem with *c* classes, is represented by the binomial distribution (Fleming & Nellis, 1994):

$$P(\mathbf{y}|\mathbf{x}\wedge\mathcal{N}) = \prod_{i=1}^{c} a_i^{y_i}(1-a_i)^{1-y_i} \tag{8-11}$$

Applying logarithm transform, the above equation becomes:

$$\begin{aligned}\mathcal{F} &= \ln P(\mathbf{y}|\mathbf{x} \wedge \mathcal{H}) \\ &= \sum_{i=1}^{c} (y_i \ln a_i + (1-y_i)\ln(1-a_i))\end{aligned} \tag{8-12}$$

Thus the problem becomes to maximise $\mathcal{F}$. By differentiating (8-12) with respect to decision action a_i, we then obtain:

$$\frac{\partial \mathcal{F}}{\partial a_i} = \frac{y_i - a_i}{a_i(1-a_i)} \tag{8-13}$$

To obtain the stationary points of $\mathcal{F}$, we set (8-13) to zero. We then have:

$$y_i - a_i = 0 \tag{8-14}$$

Since to the left of the above stationary point, $\frac{\partial \mathcal{F}}{\partial a_i}$ is positive and to the right $\frac{\partial \mathcal{F}}{\partial a_i}$ is negative, the stationary point from (8-14) is the maximum point of $\mathcal{F}$.

Since for an RBFN:

$$\hat{y}_i(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b_i \tag{8-15}$$

As discussed in Section 8.2.2, $b_i \in [0,1]$, and the classifier output as in (8-15) satisfies $\hat{y}_i \to b_i$ for an input pattern located far from the training patterns. To satisfy (8-14), we wish a_i to coincide with the actual y_i when τ_i (assuming, here, that each output node of the RBFN is allowed to have a different threshold) is applied to $\hat{y}_i$, as in (8-2). In other words, a_i should be 1 if the input pattern is

within the training class and 0 if it is out of the training class. Therefore, we must select:

$$\tau_i = b_i \tag{8-16}$$

The threshold with the value given by (8-16) decides whether samples are classified into that class. If there is noise or disturbance in the data, we expect that the turning point (where classification error rate increases abruptly) will be less sensitive to the dataset, and we therefore increase (8-16) by a small amount, ε:

$$\tau_i = b_i + \varepsilon \tag{8-17}$$

The introduction of ε is to make the classifier robust to noise and disturbance while having little increase on the misclassification rate.

Finally, if we use a single-threshold RBFN classifier, then we obtain:

$$\tau = \max(\mathbf{b}) + \varepsilon \tag{8-18}$$

Here, ε is a very small positive constant to make τ slightly greater than max(**b**): of course, τ must not exceed 1.

8.2.4 Summary of the technique

Based on an analysis of the decision behaviour of an RBFN classifier, a technique for threshold determination was derived above.

This method assumes that the RBFN classifier has been successfully trained. Following such training, an appropriate threshold value may be determined by considering values larger than the maximum value of the bias value in the output layer.

8.3 Empirical tests

In this section, the threshold determination technique derived in Section 8.2.3 is assessed in two empirical studies. The chosen datasets were obtained, firstly, from a mathematical model simulating static fault diagnosis and, secondly, from a non-linear model of a diesel engine cooling system. Both datasets were described in Chapter 5. In both cases, the input variables were normalised to [0, 1], and the classifiers were trained using the orthogonal least square algorithm (Chen, *et al*, 1991) in the Matlab Neural Network Toolbox.

As discussed in Chapter 2, in addition to determining the threshold value, implementing an effective RBFN classifier for a given task involves determining two further important parameters:

(a) the maximum number (*me*) of radial basis functions to use in the hidden layer;

(b) the spread constant (*sc*) of the radial basis function.

For each of the following two experiments, the classifier was trained using a range of possible values for *me* and *sc*. The trained classifier was then tested both on the

(seen) training set and on the (unseen) test sets. The classification error on each dataset is estimated using (8-5).

The key purpose of each study was to explore the impact of the threshold value. To this end a 'traditional' threshold value (τ_0) of 0.5 was used. This was compared with a threshold value (τ_1) determined according to (8-18). More explicitly, ε was selected as:

$$\varepsilon = 0.05 \times \max(\mathbf{b})$$

to provide a threshold slightly grater than max(**b**).

8.3.1 Mathematical model dataset

This mathematical model representing a class of static fault diagnosis problems is described in Chapter 5 (Section 5.3). Using that model, one set of training data was generated with values of p_1 and p_2 sampled from the normal distribution $N(0, 0.25)$ and $v_1, v_2 \sim N(0,0.015)$ In total, 600 input/output pairs were generated from (8-20) and were used for training all the networks.

Two additional sets of test data, each with 300 input/output pairs, were also generated, designated 'Test Set 1' and 'Test Set 2'. These datasets were intended to explore how our approach performs, both in terms of interpolation (Test Set 1) and extrapolation (Test Set 2).

Test Set 1 had the same distribution as the training set. Test Set 2 had values distributed over the whole parameter space. For this set, samples within the region of training data were assigned to one of the known classes, all other samples were assumed to belong to unknown faults.

Using the training dataset, RBFN classifiers were trained by changing the number and the spread constant of radial basis functions in the hidden layer. Table 8-1 lists the misclassification rate of the trained RBFN classifiers. In the table: training data, Test Set 1 and Test Set 2 are represented by $\mathcal{D}_0$, $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively; elements in row **b** are biases of the corresponding output neurons; Rows 5 and 6 are the misclassification rates for the training sets using τ_0=0.5 and τ_1=max(**b**)+ε, respectively; Rows 6 to 9 are misclassification rates for Test Set 1 and Test Set 2. As noted above, we set $\varepsilon = 0.05 \times \max(b)$.

Classifier Number	1	2	3	4	5	6	7	8	9	10	11
me/sc	50/0.025	50/0.03	50/0.05	50/0.1	100/0.025	100/0.04	100/0.05	100/0.06	100/0.075	100/0.1	100/0.2
b	0.0224 0.5684 0.4092	0.1230 0.4196 0.4574	-0.0045 0.4616 0.5429	0.0065 0.3969 0.5966	0.0629 0.5021 0.4350	0.0154 0.5801 0.4045	-0.0027 0.6372 0.3655	0.0283 0.5832 0.3885	0.0100 0.3828 0.6072	-0.0058 0.3922 0.6137	-0.4157 0.6168 0.7990
$\mathcal{D}_0$ with 0.5	4.67	5.17	0.83	0.5	0.5	0.33	0.33	0.33	0.33	0.33	0.5
$\mathcal{D}_0$ with τ_1	11.83	4.17	1.0	1.17	0.67	0.5	0.33	0.33	0.5	0.5	6.5
$\mathcal{D}_1$ with 0.5	9.67	9.0	2.33	1.67	2.67	1.33	0.67	1.0	1.33	2.0	3.67
$\mathcal{D}_1$ with τ_1	23.67	8.0	3.0	2.67	4.67	2.0	2.67	1.33	2.67	3.0	15.67
$\mathcal{D}_2$ with 0.5	53	9.33	47.0	45.33	48.33	47.67	47.67	47.0	47.67	47.33	52.0
$\mathcal{D}_2$ with τ_1	14.67	8.67	7.67	22.33	7.67	5.67	7.67	9.67	15.33	27.33	54.33

Table 8-1. Classification error rate (%) for mathematical model.

From the table, it is clear that when using either τ_0 or τ_1, the classifiers provide a very similar misclassification rate on the training set and test set 1. However, for samples out of the training region the classifier using τ_1 produces a lower misclassification rate.

These results may be readily understood. They arise because the appropriately trained classifier is intended to produce a high output value (close to 1) for samples in the class while a low output value (close to 0) for samples in other classes. Thus there is a large interval for threshold selection. Theoretically one could use any threshold between 0 and 1 for a perfectly-trained classifier which is required only to classify samples within the 'training' range.

To further explore the effects of the value of the threshold on the misclassification rate, different thresholds were used for the classifiers in Table 8-1 with a maximum of 100 hidden neurons and spread constant of 0.025 and 0.04. Figure 8-2 and Figure 8-3 show the misclassification rate versus the threshold.

Table 8-2 lists the misclassification rates versus thresholds around max(**b**). Here ε in (8-18) was set to be proportional to max(**b**), that is, $\varepsilon = \lambda \cdot \max(\mathbf{b})$, where λ is a small coefficient in {-0.1, -0.05, -0.01, 0.0, 0.01, 0.05, 0.1}.

These empirical results confirm the findings in (8-18). Specifically, as is apparent in Figure 8-2 and Figure 8-3, the classifier with a threshold slightly larger than

max(**b**), say τ_1, produces a minimum misclassification rate for Test 2 while a near minimum misclassification rate for the training set and Test Set 1, and τ_1 is the turning point of misclassification rate on Test Set 2. This is because, if the threshold is smaller than τ_1, the classifier will misclassify samples out of the training range as existing known classes. If the threshold is bigger than τ_1, the classifier will reject more samples within known classes, and hence increase the misclassification rate for the training set and Test set 1. For the former case, the classifier assigns an unknown class to known class(es), and may assign 'unknown fault' to 'normal condition'.

It is frequently a requirement (in CMFD applications) that the classifier will provide us not only with a high performance for known conditions but also good performance in the presence of unknown faults. For this purpose, these empirical results support the use of a classifier with threshold τ_1. Moreover, the determination of τ_1 is straightforward from the biases in the output layer. This avoids the risks inherent in an arbitrary selection of the threshold value.

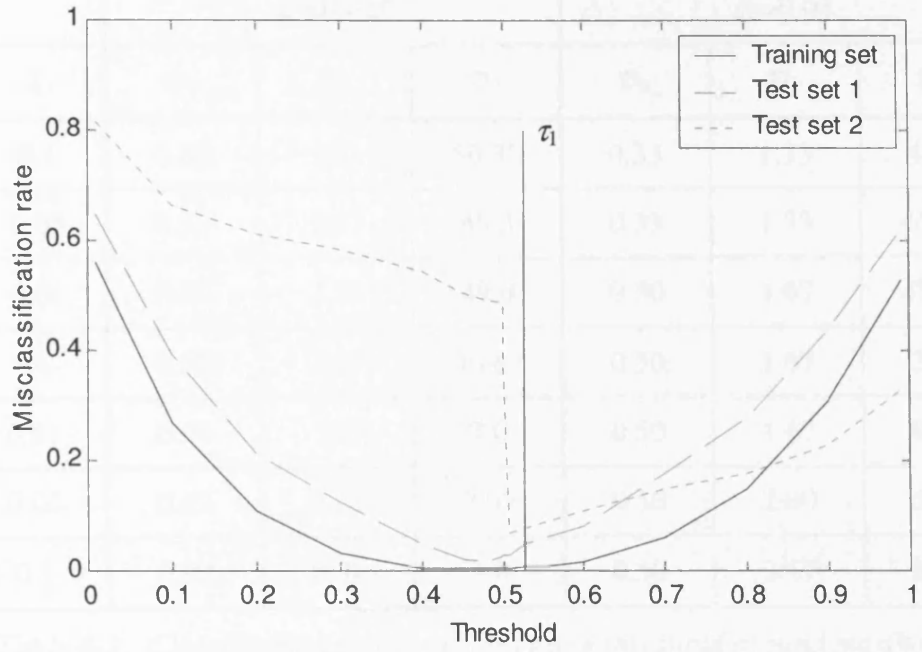


Figure 8-2. Misclassification rate vs. Threshold (sc=0.025).

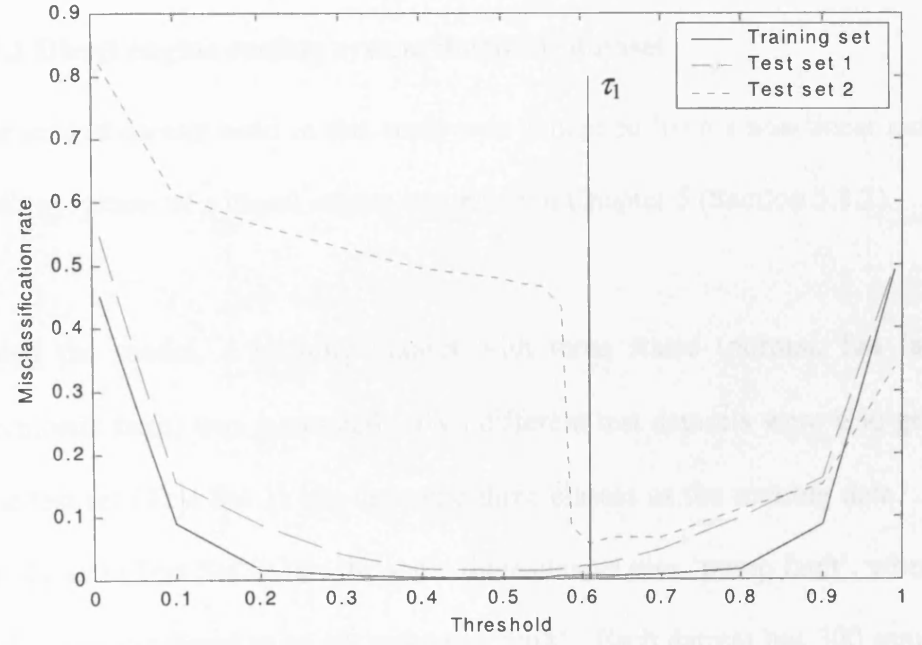


Figure 8-3. Misclassification rate vs. Threshold (sc=0.04).

	sc=0.025			sc=0.04		
λ	$\mathcal{D}_0$	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_0$	$\mathcal{D}_1$	$\mathcal{D}_2$
-0.1	0.33	2.0	50.33	0.33	1.33	47.0
-0.05	0.33	1.67	49.3	0.33	1.33	46.33
-0.01	0.50	2.33	49.0	0.50	1.67	43.33
0	0.50	2.67	36.67	0.50	1.67	29.0
0.01	0.50	3.0	7.0	0.50	1.67	8.33
0.05	0.67	4.67	7.67	0.50	2.00	5.67
0.1	0.83	6.0	9.0	0.50	2.67	6.67

*Table 8-2. Classification error rate (%) using threshold around max(**b**)*
$\tau = \max(\mathbf{b}) + \varepsilon = \max(\mathbf{b}) + \lambda \cdot \max(\mathbf{b})$.

8.3.2 Diesel engine cooling system diagnosis dataset

The second dataset used in this study was generated from a non-linear model of a cooling system of a diesel engine described in Chapter 5 (Section 5.3.2).

Using the model, a training dataset with three states (normal, fan fault, and thermostat fault) was generated. Two different test datasets were also generated. One test set (Test Set 1) has the same three classes as the training data. Another test dataset (Test Set 2) has the same three classes plus 'pump fault', where pump fault was considered to be an 'unknown fault'. Each dataset has 300 samples. In each case, the datasets consisted of equal numbers of samples for each class, that is, each class has 75 samples in Test Set 2, and 100 samples in the training set and Test Set 1.

Classifier Number	1	2	3	4	5	6
me/sc	100/0.15	100/0.2	100/0.25	100/0.26	100/0.27	100/0.3
MSE	0.1799	0.1147	0.0962	0.0934	0.01090	0.0885
b	0.3080 0.3451 0.3469	0.1825 0.4873 0.3302	0.0898 0.4941 0.4161	0.1583 0.6440 0.1978	0.1341 0.6847 0.1812	0.0527 0.7573 0.1900
$\mathcal{D}_0$ with 0.5	15.67	8.33	5.33	5.0	7.67	5.33
$\mathcal{D}_0$ with τ_1	18.67	8.67	6.00	12.33	20.0	22.33
$\mathcal{D}_1$ with 0.5	21.67	13.0	10.33	11.33	11.0	10.33
$\mathcal{D}_1$ with τ_1	22.33	13.67	10.33	18.0	20.67	21.67
$\mathcal{D}_2$ with 0.5	20.0	12.67	12.0	35.67	35.33	35.33
$\mathcal{D}_2$ with τ_1	18.33	13.33	11.67	17.0	19.67	20.67

Table 8-3. Classification error rate (%) for cooling system diagnosis.

To explore the effects of the value of the threshold on the misclassification rate, we used different thresholds for the classifier in Table 8-3 with a maximum number of hidden neurons of 100 and spread constants of 0.25 and 0.27. Figure 8-4 shows the misclassification rate versus the threshold. Table 8-4 lists the misclassification rates versus thresholds around max(**b**).

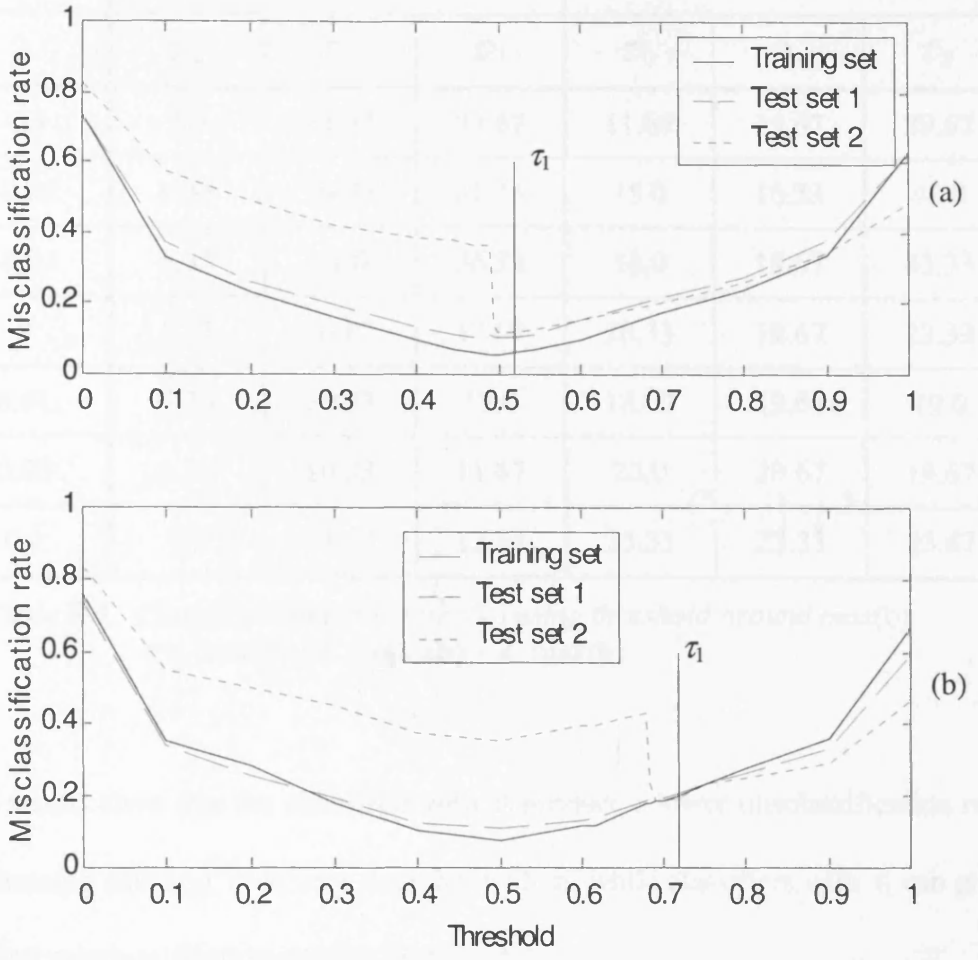


Figure 8-4. Misclassification rate vs. threshold (a) for sc=0.25 and (b) for sc=0.27.

	sc=0.25			sc=0.27		
λ	$\mathcal{D}_0$	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_0$	$\mathcal{D}_1$	$\mathcal{D}_2$
-0.1	7.0	11.33	37.67	11.67	13.67	39.67
-0.05	6.33	10.33	36.33	15.0	16.33	42.0
-0.01	5.33	11.0	36.33	18.0	18.67	43.33
0	5.33	10.67	13.67	18.33	19.67	23.33
0.01	5.33	10.33	12.0	18.67	19.67	19.0
0.05	6.33	10.33	11.67	20.0	20.67	19.67
0.1	7.0	11.33	12.33	23.33	23.33	23.67

*Table 8-4. Classification error rate (%) using threshold around max(*b*)*
$\tau = \max(\mathbf{b}) + \varepsilon = \max(\mathbf{b}) + \lambda \cdot \max(\mathbf{b})$.

The results show that the classifiers with τ_0 produce a lower misclassification rate for training data and Test Set 1 than that with τ_1, while classifiers with τ_1 can give a better misclassification rate for Test Set 2.

As demonstrated in Experiment 1, the classifier with τ_1 gives near minimum misclassification rate for samples within the range of training data, so τ_1 can still be used in such applications. If we are concerned with the classifier performance in applications with possible unknown classes, τ_1 is more suitable but incurs some cost in misclassification of known classes.

8.4 Discussion

The proposed approach has been tested using two classification problems: static fault diagnosis and cooling system diagnosis. Each classifier was trained using a training dataset ($\mathcal{D}_0$), and then tested by a Test Set 1 ($\mathcal{D}_1$) which has the same class distribution as the training data. To compare the proposed approach with traditional approaches, we also tested the classifiers for the problems using an additional test dataset ($\mathcal{D}_2$) which has a new class not seen by the classifier during training. For fault classification applications, the new class represents an unknown fault.

The experimental results demonstrated that, on $\mathcal{D}_0$ and $\mathcal{D}_1$, the proposed approach gives a sub-optimal threshold for both of the two experiments, and the classifiers with τ_1 produce near minimum misclassification rate. However on $\mathcal{D}_2$ the performance of classifiers with τ_1 was significantly improved, and reached a minimum misclassification rate.

The value of the threshold affects the classifier performance in two ways. On the one hand, in most cases, if τ_1 is bigger than τ_0 the classifier rejects more samples in $\mathcal{D}_0$ and $\mathcal{D}_1$ and thus increases the misclassification rate. On the other hand, a higher confidence in the classification of the samples is obtained.

It should be noted that it has also been shown that the bias, b, at the output layer of an RBFN classifier should satisfy the condition $b \in [0,1]$. This result may be

used to check whether a particular classifier has been trained successfully, and the classifier should be discarded if max(**b**)>1 or min(**b**)<0. Based on this condition, it would be sensible to abandon classifiers numbered 3, 7, 10 and 11 in Table 8-1.

Finally, note that care must be taken when accepting the training if max(**b**) is close to 1, because the classifier will leave little margin for response to previously unseen samples. An 'ideal' RBFN classifier will therefore have a low misclassification rate and a small value for max(**b**).

8.5 Conclusions

An approach for determining a reliable threshold for RBFN classifiers has been derived. This approach is easy to use and understand.

The proposed approach is especially useful in classification problems where there may be possible new classes or 'unknown faults'. The result obtained when testing the approach with two such classification problems used in this chapter demonstrated the effectiveness of this technique.

9
A METHODOLOGY FOR DESIGNING EMBEDDED CMFD SYSTEMS

9.1 Introduction

Throughout the previous chapters, various techniques intended to build effective classification systems for embedded CMFD applications have been presented. The building of CMFD classification systems usually consists of three stages: pre-processing, classifier design and post-processing. Therefore the techniques have been developed for achieving more effective building components.

The techniques for pre-processing and post-processing were developed theoretically and assessed by a series of experiments. The classifier design, though, was mainly evaluated empirically and, where possible, subject to the appropriate theoretical analysis. Based on the theoretical investigation and the empirical results, this chapter produces a classification system design methodology to assist in the effective use of MLP and RBFN classifiers in embedded CMFD applications. In order to assess this methodology, it is applied to the design of an embedded CMFD system for fault diagnosis in the aspiration system of a diesel engine.

9.2 Towards a design methodology

This section seeks to draw together the results of the various experiments obtained in previous chapters, and thereby develop a preliminary design methodology: this is intended to assist in the effective use of MLP and RBFN classifiers in embedded CMFD applications. This section firstly summarises the techniques and results from the individual stages. It then presents the classification system design methodology along with a design flowchart.

9.2.1 Pre-processing strategy

The first step to design a CMFD classification system is to extract features using appropriate signal processing technique. Chapters 3 and 4 presented a separability measure based on a non-parametric analysis of the data that can be used to select an appropriate pre-processing approach:

$$J = \frac{1}{n}\mathrm{tr}(\mathbf{S}_w^{-1}\mathbf{S}_b)$$

The goal in this stage of the design process is to seek the pre-processing technique that gives the feature dataset with the largest separability. Thus a technique for identifying the most effective pre-processing approach can be summarised as follows:

1) Take the recorded raw dataset, and apply the chosen pre-processing strategy.

2) Measure the separability between classes after pre-processing.

3) Repeat for all alternative pre-processing strategies.

4) In the classifier system, employ the pre-processing strategy that results in the largest separability measure.

9.2.2 Classifier selection criteria

As discussed in Chapter 5, when designing an embedded CMFD classifier system using an MLP or RBFN classifier, three key factors must be taken into account: (1) the basic classifier performance, (2) hardware resource implications, and (3) performance when there is limited training data, and in the presence of 'unknown' or 'multiple' faults. From the results obtained in Chapters 6 and 7, it should be noted that neither MLP nor RBFN exhibits the best performance when measured against these key criteria.

To assist in the selection of the most suitable classifier, the following "rule of thumb" can be drawn from the experimental results in Chapter 6 and 7.

1) If the classification error rate in known faults is the only factor being considered, then either MLP or RBFN may be used as they provide very similar levels of performance.

2) The training of an MLP requires considerably more processor operations than that of an RBFN, while the testing of an MLP requires rather fewer processor operations than an RBFN. In cases where off-line training is possible, and rapid classification ('testing') is required, the MLP may be a more appropriate choice. However, where on-line learning is required, the RBFN may be more appropriate.

3) In the phase of classification, an MLP requires less memory due to having fewer hidden nodes and parameters (weights and bias).

4) The MLP requires fewer training samples to achieve good generalisation. Since the training speed of RBFN is faster than MLP, the MLP will require much longer training time if the training set is large. Thus, on the basis of these findings, it can be concluded that: it is more appropriate to use RBFN in situations where there are 'large' numbers of (training) samples available, and MLP in situations where only 'small' numbers of (training) samples are available. Both classifiers may be considered if the training set is of 'medium' size.

5) Both MLP and RBFN perform similarly in the presence of multiple faults, they can be considered equally in such applications.

6) RBFN can provide accurate classification results for unknown faults if they are well separated from known classes, but still (inevitably) performs poorly for those unknown faults which overlap with known classes. On the other hand, MLP is prone to classify samples of unknown faults as known classes since the decision boundaries of the known classes are unbounded. Overall where unknown faults are concerned, RBFN is the more appropriate classifier.

It is clear that there is no unique best classifier against selection criteria. The selection of the most appropriate classifier must be based on the constraints and the nature of the specific application.

9.2.3 Post-processing strategy

CMFD classification systems have their distinct features as detailed in previous chapters. These include the particular attention that should be paid to the interpretation of the classifier outputs. A threshold-based post-processing scheme was considered more appropriate for CMFD applications because of the nature of this area. The key issue of this threshold-based scheme is the determination of the threshold value. For most cases both MLP and RBFN classifiers can employ a value of 0.5 as is commonly used in traditional applications. However for CMFD applications with unknown faults, RBFN classifiers should be considered and the threshold value must be determined. Chapter 8 developed a technique that may significantly improve the performance of RBFN classifiers in CMFD applications where unknown faults may occur. As a result, the post-processing scheme can be presented as follows.

1) For problems where all classes are known *a priori*, a threshold of 0.5 is suitable for both MLP and RBFN classifiers.
2) For problems with unknown faults, RBFN should be selected. The threshold of the post-processing should be set to a value slightly greater than the maximum bias of the output layer of the RBFN classifier.

9.2.4 Overall classification system design methodology

The previous sections discussed the techniques and considerations that aim to achieve the best performance in individual design stages. In order to make the techniques accessible, this section presents the methodology for designing effective classification systems for embedded CMFD applications using MLP and

RBFN classifiers. The basic idea of the design methodology relies on the common process of CMFD systems that consist of pre-processing, classifier selection and post-processing. Thus the presented methodology intends to build CMFD systems from three reliable stages. Figure 9-1 presents the flowchart of the design methodology.

In keeping with the observations presented above, and Figure 9-1, the process of the design methodology is described as follows by integrating the previous individual strategies together.

- **Pre-processing**
    1) Take the recorded raw dataset, S, and apply the chosen pre-processing technique, SP_i.
    2) Measure the separability between classes after pre-processing, J_i.
    3) Repeat for all alternative pre-processing techniques.
    4) In the classifier system, employ the pre-processing technique that results in the largest separability measure.
- **Classifier selection**
    5) Consider the required classification error.
    6) Consider the available hardware resources (and/or system cost).
    7) Consider the available training data.
    8) Consider the need to deal with 'unknown' or 'multiple' faults.
- **Post-processing**
    9) For problems that all classes are known *a priori*, a threshold of 0.5 is applicable for both MLP and RBFN classifiers.
    10) For problems with unknown faults, an RBFN should be selected. The threshold of the post-processing is set to a value slightly greater than the maximum bias of the output layer of the RBFN classifier.

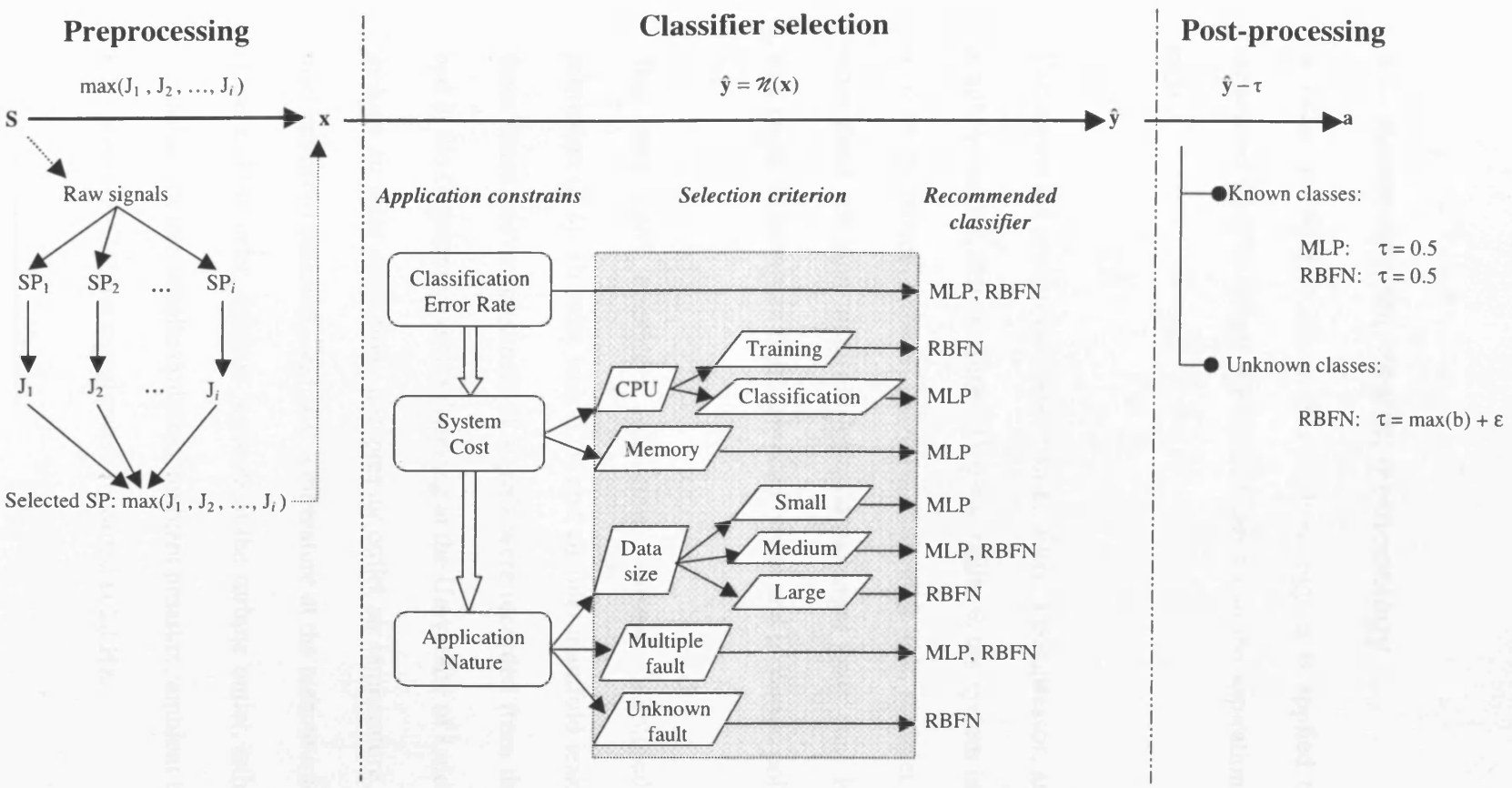


Figure 9-1. Flowchart for designing an appropriate embedded CMFD system using MLP or RBFN classifier. Keys: **S***–raw signals,* SP*–signal processing technique,* J*–separability measure,* **x***–feature vector,* $\mathcal{N}$ *–classifier,* $\hat{\mathbf{y}}$*–output of classifier,* τ*–threshold,* **a***–final decision.*

9.3 Assessing the design methodology

In order to begin to assess this methodology, it is applied to the design of an embedded CMFD system for fault diagnosis in the aspiration system of a diesel engine.

The aspiration system consists of an air filter, a compressor, an air inlet manifold, a turbine and an exhaust pipe. Possible faults in the system include exhaust leak, exhaust restriction, exhaust valve fault, air inlet leak, air inlet restriction, air inlet valve fault, air inlet manifold leak, turbocharger fault, etc. Faults in the system may result in unburned fuel, degraded power and increased pollution.

This case study considers four states: normal (identified by D1), exhaust restriction (D2), air inlet leak (D3) and air inlet manifold leak (D4). For each of these states, eleven channels of signals were recorded from the diesel engine test bed in the Department of Engineering at the University of Leicester[19]. The signals include air inlet mass flow, compressor outlet air temperature, boost pressure (air inlet manifold pressure), exhaust temperature at the turbine inlet, exhaust pressure at the turbine inlet, exhaust pressure at the turbine outlet, exhaust temperature at the turbine outlet, smoke emission, ambient pressure, ambient temperature, and air inlet pressure. All the signals were recorded at 20 Hz.

---

[19] The author would like to express his sincere thanks to Dr John Twiddle who supplied the recorded data for use in this study.

The recorded signals contained a large number of samples (most of the signals with over 25k samples). Data samples in these recorded signals were then re-organised into the form for the design of fault classifier, specifically, a training dataset (containing 600 samples, 150 samples for each of the four classes), and a test dataset (containing 1000 samples, 250 samples for each class).

9.3.1 System requirements and initial consideration

The requirements for the embedded aspiration diagnosis system were identified as follows:

1) The system should be implemented at low cost, preferably on an 8-bit microcontroller.
2) The system should work in real-time.
3) The system should provide an 'excellent' classification error rate.
4) The system should be capable of working in an 'unpredictable' environment.

Since the experiment was conducted under several steady states and all the signals changed very little under each state, the signals are directly used to form the feature vectors. This is somewhat unusual because most CMFD applications require feature extraction through pre-processing. Nevertheless this application problem is still valuable for assessing the presented design methodology because the pre-processing strategy was sufficiently evaluated using simulated and real

problems in Chapters 3 and 4, while this experiment will focus on the assessment of classifier selection and post-processing.

The design process therefore began with a consideration of the classification error. In this study, there are a very large amount of samples available for classifier design. As the results in Chapter 8 demonstrated, there is little to choose between RBFN and MLP classifiers when there is sufficient training data. This suggests that, on the basis of classification performance alone, either an MLP or RBFN classifier could be used in this case.

The next, key, consideration is one of cost. The system cost is directly related to the hardware resource requirements, such as memory and CPU load. As summarised in Figure 9-1, the results in Chapter 6 demonstrated that the MLP has lower memory requirements than RBFN. In addition, if we are only concerned with classification (and not training) then the MLP also imposes a lower CPU cost, allowing the use of a less expensive microcontroller and/or (as discussed in Chapter 6) reduced system power consumption. Together, these observations tend to suggest that an MLP solution may be the most appropriate.

However, basic performance and cost are not the only issues to be considered. As introduced in the beginning of this section, there are possibly nine (or more) condition states in the aspiration system, but only four states are considered in this

study[20]. Thus the designed CMFD system is expected to work under the environment of possible unknown states. According to Figure 9-1, a RBFN may have to be considered if unknown faults are to be detected.

On the basis of the methodology presented earlier in this chapter, the MLP should be selected (to reduce product cost and, possibly, power consumption) while RBFN should be selected, to improve the likelihood of detecting 'unknown faults'. On the basis of this analysis, a compromise solution must be made depending on which aspect of the system requirements is considered to be the most important.

In the next section, these statements are assessed empirically.

9.3.2 Experiments

A series of experiments were carried out to assess the prediction derived from Figure 9-1 as above. Figure 9-2 depicts the structure of MLP or RBFN classifiers for this engine aspiration diagnosis application.

---

[20] This is partly because, as in many CMFD systems, some of the other states - such as exhaust leak - are very difficult to produce on the testbed.

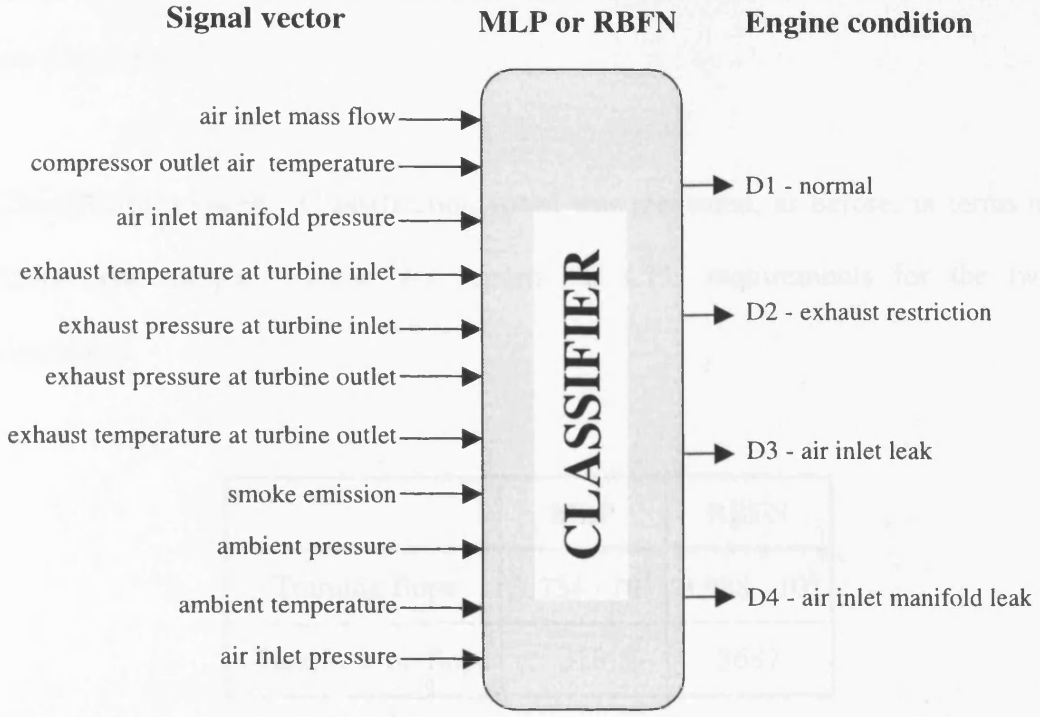


Figure 9-2. Structure of the classifiers for engine aspiration system diagnosis.

Classification error rate. The classifiers were trained on the training dataset with 600 samples, and were then tested on 1000 unseen samples (the datasets were described in the beginning of Section 9.3). Table 9-1 lists the error rate (in percent).

	MLP	RBFN
Hidden nodes	8	49
Training error (%)	0.5	0.2
Test error (%)	10.6	10.2

Table 9-1. Classification error rate for Engine aspiration system diagnosis.

As predicted in Section 9.2.4, the classification error rates of the two classifiers are very similar.

Classification speed. Classification speed was measured, as before, in terms of 'flops' per sample. Table 9-2 reports the CPU requirements for the two classifiers.

	MLP	RBFN
Training flops	$7.734 \cdot 10^9$	$3.988 \cdot 10^8$
Classification flops	328	3687

Table 9-2. CPU requirements for Engine aspiration system diagnosis.

As stated in Section 9.2.4, the classification speed of the MLP is much faster than that for the RBFN.

Memory requirement. The classifiers were implemented on 8-bit microcontrollers, which are the mostly common used and can be embedded in most plants as well as diesel engines. The embedded C source code for these classifiers is listed in appendix.

Looking at the C source code, only the essential parts of the classification process are implemented in these experiments. Since the classifiers are trained before embedding on microcontrollers, the parameters of the classifiers are assumed to be fixed and stored in ROM.

Table 9-3 shows the memory requirements of these classifiers. From the table, it is seen that the MLP classifier requires much less memory (~50% less) than the RBFN.

	MLP	RBFN
8-bit	9437	17378

Table 9-3. Memory requirement on 8-bit microcontroller for Engine aspiration system diagnosis.

Detection of unknown faults. In this experiment, the training dataset with 600 samples contains three states: normal (D1), exhaust restriction (D2) and air inlet leak (D3). The test dataset with 1000 samples consists of four states: D1, D2, D3 and air inlet manifold leak (D4). D4 is considered to represent one of the unknown faults that may be encountered in the real operating environment.

The confusion matrix of MLP and RBFN on the test dataset is presented in Table 9-4 and Table 9-5. It is observed that both MLP and RBFN provide similar classification performance for the known classes (D1, D2, and D3). However MLP misclassifies 98.8% of D4 as D2, so it is unable to identify the unknown class D4. By contrast, the RBFN correctly detects the unknown class D4 (73.6% accuracy), and only comparatively rarely misclassifies D4 as D1 (7.6%) or as D2 (18.8%).

	D1	D2	D3	D4
D1	0.74	0	0.168	0.092
D2	0	0.992	0	0.008
D3	0	0	0.992	0.008
D4	0	0.988	0	0.012

Table 9-4. Confusion matrix of classification (MLP).

	D1	D2	D3	D4
D1	0.736	0	0.012	0.252
D2	0	0.988	0	0.012
D3	0	0	0.972	0.028
D4	0.076	0.188	0	0.736

Table 9-5. Confusion matrix of classification (RBFN).

Overall, the results of these experiments confirm the earlier findings, and support the design methodology summarised in Figure 9-1.

9.4 Conclusions

On the basis of the results obtained in these studies, it is argued that each form of classifier has both strengths and weaknesses, and that neither is suitable for use in all CMFD applications. In order to assist in the selection between MLP and RBFN classifiers, a design methodology for CMFD applications was proposed based on the results. An assessment of the design methodology on a new case study confirmed its value when selecting between MLP and RBFN classifiers for use in embedded CMFD applications.

10
CONCLUSIONS

10.1 Introduction

The aim of the programme of work described in this thesis was to investigate how MLPs and RBFNs techniques could be most effectively applied in embedded CMFD applications. The particular focus of the work was on what are referred to here as "three-stage neural classifiers"; such classifiers involve the pre-processing of raw data from the plant, the design of suitable (neural) classifiers, and the post-processing of classifier outputs. In such a three-stage system, each stage contributes significantly to the system performance.

In this chapter, the results obtained are discussed, and the extent to which the aims of the thesis have been achieved is considered. In addition, some suggestions for future work in this important area are made.

10.2 Techniques for effective pre- and post-processing

The thesis explored two techniques through which the performance of the chosen classifier could be improved.

The first area considered involved the process of selecting pre-processing strategies for embedded classifiers. As discussed in Chapters 3 and 4, the 'raw' signals obtained from sensors are rarely applied directly to the classifier. Instead, these signals are pre-processed in order to reduce the data size, and - ideally - to emphasise relevant data features.

A variety of pre-processing techniques exist. However, as with the selection of appropriate classifier, the selection of an appropriate pre-processing method has traditionally been based on 'trial-and-error'. As an alternative, Chapter 4 presented a strategy for selecting pre-processing methods which is based on a separability matrix. The matrix was derived from a non-parametric analysis of classes in the dataset: crucially, it requires no assumptions to be made about the underlying distribution of the data. Since the data distributions for practical CMFD applications are likely to be unknown *a priori*, the proposed separability analysis is particularly useful for CMFD applications. The problem of engine misfire detection was used to demonstrate the effectiveness of this technique.

Another contribution made in this thesis concerned the selection of thresholds for RBFN classifiers. Again, the focus was on CMFD applications, particularly those with multiple faults or unknown faults. Following an investigation of the underlying theory, a technique for deriving the required threshold for a trained RBFN classifier was developed. This was then assessed empirically in two further case studies.

10.3 Comparing classifier performance

Chapters 6 and 7 of this thesis presented the results of a comprehensive series of empirical studies aimed at comparing the performance of MLP and RBFN classifiers. As introduced in Chapter 1, several benchmark and comparison studies have been previously published which have compared the performance of different classifier systems (Ripley, 1994; Jain & Mao, 1997; Yang, 2000). However, five particular problems distinguish many embedded CMFD applications from most generic classifiers:

1) CPU resources are likely to be limited.
2) Memory resources are likely to be limited.
3) 'Multiple faults' can occur.
4) 'Unknown faults' can occur.
5) Limited training data may be available.

The studies of classifiers discussed in this thesis are, it is believed, unique in their focus on the design of embedded CMFD systems. Table 7-1 summarises the results obtained.

Rule	Rank	Remarks
Error rate	MLP ~ RBFN	For interpolation
Training speed	RBFN >> MLP	
Classification speed	MLP >> RBFN	
Memory requirement	MLP > RBFN	On microcontroller
Training data size	MLP > RBFN	
Multiple faults	MLP ~ RBFN	
Unknown faults	RBFN >> MLP	

Table 10-1. Comparison Results. In the table the relative order of classifier A and classifier B is represented as follows.
A ~ B: classifiers A and B have similar performance.
A > B: classifier A is slightly better than classifier B.
A >> B: classifier A is clearly better than classifier B.

10.4 Can the comparison results be generalised?

The results obtained in previous chapters were based mainly on the results of empirical studies, using a limited number of datasets and data samples. It is therefore important to consider the extent to which these results may be generalised. Two observations are made below.

10.4.1 Selection of datasets

As classifiers can be applied to many areas in science and technology, the number of datasets from possible domains is very large and the differences among them is great. It is clearly impossible to compare classifiers by using them on all possible applications.

To help address this problem, this thesis did not directly consider the physical representation of the dataset, but instead focused on the data characteristics. A dataset can be characterised by a number of measures, one of the important measures is the separability between classes. To make the measure distribution-free, this thesis used a non-parametric separability matrix to characterise the dataset. Datasets from three case studies were then chosen that posed different levels of difficulty for the classifier. For example, some classes had non-linear boundaries (in the MM study), some classes were well separated (normal, pump fault and radiator fault in the DC study), some classes had small overlaps and complex class boundaries (in the BC study), some classes overlapped strongly (normal and thermostat stuck open in the DC study).

Overall, this range of data characteristics was felt to be highly appropriate for the type of application considered in this thesis.

10.4.2 Rules independent of data

It should also be noted that some of the results are largely independent of the particular datasets used: in this category are the processor and memory requirements. These are determined mainly by the network architecture (number of inputs, outputs and hidden units).

10.5 The design methodology

To make it easier to apply the results obtained from this study, they were used to derive a design methodology (described in detail in Chapter 9). This methodology was further explored in an additional case study, in Chapter 9. The results from this study confirmed the predictions from the methodology.

10.6 Future work

As this thesis draws to a close, some suggestions for future work in this important area are made.

10.6.1 Novelty detection

When designing classifiers, it is often assumed that the states in the plant are exhaustively known. This can be a significant drawback in real world CMFD applications where it is difficult (if not impossible) to model all the possible fault states of the plant in advance. It is therefore highly desirable that a classifier can detect plant states which were unknown *a priori.*

Recently novelty detection has become increasingly important in many different fields (Roberts, 1999; Tarassenko, *et al*, 2000). In novelty detection, a classifier is constructed from a training dataset and novel classes are consequently identified by examining the classifier output against output patterns for known classes. Novelty detection can be achieved by density estimation (Bishop, 1994; Roberts, 1999; Desforges, *et al*, 1998), neural networks (for example, RBFN and support

vector machines: Stitson *et al*, 1996), hidden Markov models (Smyth, 1994) or data fusion techniques.

The decision as to whether an input vector is from novel class or not tends to be made on the basis of an exceeded threshold. As yet, there is no principled way to choose such thresholds. The threshold selection method developed in Chapter 8 tackled this difficulty successfully for trained RBFN classifiers. It would be interesting to explore the impact of extending the technique discussed in Chapter 8 into the area of novelty detection.

10.6.2 Time-varying distributions

Suppose at time t_0 some information is obtained (in the form of measurement data and heuristic knowledge) from the system, and a classifier is constructed and trained from this information. After training, the classifier will have decision boundaries representing the states of the system at t_0. Conventionally, such boundaries of classes are assumed to be fixed. In practice, however, the actual boundaries may vary in position, orientation or shape with respect to time, due to ageing and changes in the operating environment (Martinez, 1998). It is therefore often necessary to design a classifier that can adapt its decision boundaries, from system history information (at time t_1, t_2,..., t_{n-1}), to track such variants. Thus the classifier should always provide an appropriate classification at any future time t_n, no matter how the actual boundaries vary.

Again, it seems likely that further work on threshold adjustments could be used to improve the performance in 'aging' systems.

10.6.3 Selection of training data

A neural classifier comprises a set of parameters (weights and biases) which establish relationships between the relevant system inputs and outputs. These parameters will generally be derived through training.

As discussed in Chapters 3 and 7, the quality and size of the available data has an impact on the classifier performance. In particular, in practical applications, the dataset may have a lot of redundant samples (Hara & Nakayama, 1998). On the other hand, new examples with information about the variance of class boundaries should be added into dataset to adapt the classifier.

If we wish to reduce training times, and optimise classifier performance, training data selection is required. The samples must be selected so as to maximise their information content. Lee & Landgrebe (1997) have demonstrated that samples around class boundaries contain all the necessary information for classification. Again, by adjusting the thresholds of an RBFN classifier, we can infer the position of a given sample relative to the class boundary. Taking this into consideration, some further work is justified in to the selection of training data based on the findings in Chapters 4 and 8.

10.7 Conclusions

The major contributions of the work described in this thesis fall into three areas: the predication of the effectiveness of pre-processing strategies, the comprehensive comparison for the selection of classifiers, and the determination of thresholds for optimal post-processing for RBFN classifiers. The resulting design methodology, derived from theoretical and empirical findings, was shown to be useful for designing and implementing effective embedded CMFD systems. This thesis, in answering the questions that were posed at the outset, has prompted many more. These will hopefully provide the stimulus for further research in this important area.

APPENDIX: EMBEDDED C SOURCE CODE

Embedded C code for classification using MLP

```
/* MLP classifier: essential code only */
#include <math.h>

#define InNo 12          // number of input nodes
#define HiNo 8           // number of hiddenn nodes
#define OutNo 4 // number of output nodes

// function for MLP classifier
void mlp(float* xx, float* yy, float* w1, float* b1, float* w2, float* b2)
{
    float HiOut[HiNo];
    register int i, j, k;

    for(i=0;i<HiNo;i++)
    {
        HiOut[i]=0;
            for(j=0;j<InNo;j++)
            HiOut[i] += ((xx[j]*w1[j*HiNo+i]));
        HiOut[i] += b1[i];
            HiOut[i]=1/(1+exp(-HiOut[i]));
    }

    for(k=0;k<OutNo;k++)
    {
        yy[k]=0;
            for(i=0;i<HiNo;i++)
            yy[k] += ((HiOut[i]*w2[i*OutNo+k]));
        yy[k] += b2[k];
            yy[k]=1/(1+exp(-HiOut[k]));
    }
}


int main()
{
    float x[InNo], y[OutNo];          // input and output array for one sample
       float w1[InNo][HiNo], b1[HiNo]      // weights and biases between input
                                            and hidden layer
       float w2[HiNo][OutNo], b2[OutNo];// weights and biases between hidden
                                            and //output layer

    // call mlp() for classifying the sample x, result stored in y
    mlp(x, y, w1[0], b1, w2[0], b2);

    return 0;
}
```

Embedded C code for classification using RBFN

```
/* RBFN classifier: essential code only */
#include <math.h>

#define InNo 12          // number of input nodes
#define HiNo 49          // number of hidden nodes
#define OutNo 4 // number of output nodes

// classification function
void rbf(float* xx, float* yy, float* Cen, float* Var, float* w2, float* b2)
{
    float HiOut[HiNo];
    register int i, j, k;

    for(i=0;i<HiNo;i++)
    {
        HiOut[i]=0;
            for(j=0;j<InNo;j++)
            HiOut[i] += ((xx[j]-Cen[j*HiNo+i])*(xx[j]-Cen[j*HiNo+i]));
        HiOut[i] /= (2.0*Var[i]*Var[i]);
            HiOut[i]=exp(-HiOut[i]);
    }


    for (k=0;k<OutNo;k++)
    {
        yy[k]=0;
            for (0;i<HiNo;i++)
            yy[k] += HiOut[i]*w2[i*OutNo+k];
            yy[k] += b2[k];
    }
}

int main()
{
    float x[InNo], y[OutNo];   // input and output array for one sample
    float cen[InNo][HiNo];     // centres of RBFN
    float var[HiNo];           // widths of RBFN
    float w2[HiNo][OutNo];     // weights
    float b2[OutNo];           // biases

    // call rbf() for classifing the sample x, the result is stored in y
    rbf(x, y, cen[0], var, w2[0], b2);

    return 0;
}
```

BIBLIOGRAPHY

Ayoubi M, Isermann R (1997). "Neuro-fuzzy systems for diagnosis," Fuzzy Sets and Systems, 89(3): 289-307.

Azzoni P, Moro D, Ponti F, Rizzoni G (1998). "Engine and load torque estimation with application to electronic throttle control," SAE Special Publications, vol.1357: 149-159.

Bartal Y, Lin J, Uhrig RE (1995). "Nuclear-power-plant transient diagnostics using artificial neural networks that allow don't-know classifications," Nuclear Technology, 110(3): 436-449.

Baum EB, Haussler D (1989). "What size net gives valid generalization?" Neural Computation, 1(1): 151-160.

Bishop CM (1994). "Novelty detection and neural-network validation," IEE Proceedings-Vision Image and Signal Processing, 141(4): 217-222.

Bishop CM (1995). "Neural networks for pattern recognition," Clarendon Press, Oxford.

Blayo F, Cheneval Y, *et al* (1995). "Enhanced learning for evolutive neural architecture," FTP: ftp.dice.ul.ac.be/pub/neural-nets/ELENA/Benchmarks.ps.Z.

Braun S (1986). "Mechanical signature analysis : theory and applications," London : Academic Press.

Broomhead DS, Lowe D (1988). "Multivariable function interpolation and adaptive networks," Complex Systems, 2: 321-335.

Bugmann G (1998). "Normalised Gaussian radial basis function networks," Neurocomputing, 20(1-3): 97-110.

Calcutt DM (1998). 8051 Microcontrollers : Hardware, Software and Applications. London: Arnold

Carelse XF (2002). "An introduction to the industrial applications of microcontrollers," Physica Scripta, T97: 148-151.

Carley L (1997). "Understanding OBDII: past, present & future," http://members.aol.com/carpix256/library/us796obd.txt.

Ceccarani M, Rebottini C, Bettini R (1998). "Engine misfire monitoring for a V12 engine by exhaust pressure analysis," SAE Special Publications, vol.1357: 65-70.

Chan WL, Chan TM, Pang SL, So ATP (1997). "A distributed on-line HV transmission condition monitoring information system," IEEE Transactions on Power Delivery, 12(2): 707-713.

Chantler MJ, Coghill GM, Shen Q, Leitch RR (1998). "Selecting tools and techniques for model-based diagnosis," Artificial Intelligence in Engineering, 12(1-2): 81-98.

Chen S, Cowan FN, Grant PM (1991). "Orthogonal least squares learning algorithm for radial basis function networks," IEEE Transactions on Neural Networks, 2(2): 302-309.

Cheng B, Titterington DM (1994). "Neural networks: a review from a statistical perspective," Statistical Science-a Review Journal of the Institute of Mathematical Statistics, 9(1): 2-30.

Cheon SW, Chang SH, Chung HY, Bien ZN (1993). "Application of neural networks to multiple alarm processing and diagnosis in nuclear-power-plants," IEEE Transactions on Nuclear Science, 40 (1): 11-20.

Chin H, Danai K (1991). "A method of fault signature extraction for improved diagnosis," Transactions of the ASME, J. of dynamic systems, measurement, and control, 113(4): 634-638.

Chung HY, Bien ZN, Park JH, Seong PH (1994). "Incipient multiple-fault diagnosis in real-time with application to large-scale systems," IEEE Transactions on Nuclear Science, 41(4 Pt2): 1692-1703.

Connolly FF, Rizzoni G (1994). "Real time estimation of engine torque for the detection of engine misfires," Journal of dynamic systems, measurement, and control, 116: 675-686.

Cordella LP, Destefano C, Tortorella F, Vento M (1995). "A method for improving classification reliability of multilayer perceptrons," IEEE Transactions on Neural Networks, 6 (5): 1140-1147.

Dash S, Venkatasubramanian V (2000). "Challenges in the industrial applications of fault diagnostic systems," Computers & Chemical Engineering, 24(2-7): 785-791.

Davies P, Silverstein BR (1995). "A comparison of neural nets to statistical classifiers for stubborn classification problems," ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, vol.5: 3467-3470.

DeJong RG, Powell RE, Manning JE (1986). "Engine monitoring using vibration signals," International off-highway & powerplant congress & exposition, (Conf. Code, 08833), Milwaukee, WI, USA, p7.

Desforges MJ, Jacob PJ, Cooper JE (1998). "Applications of probability density estimation to the detection of abnormal conditions in engineering," Journal of Mechanical Engineering Science, 212(8): 687-703.

Duin RPW (1995). "Small sample size generalization," in: G. Borgefors (eds.), SCIA'95, Proc. 9th Scandinavian Conf. on Image Analysis, (Uppsala, Sweden), vol.2: 957-964.

Duin RPW (1996). "A note on comparing classifiers," Pattern Recognition Letters, 17(5): 529-536.

Efron B (1979). "Bootstrap methods: another look at the Jackknife," Annals of Statistics, 7: 1-26.

Efron B (1983). "Estimating the error rate of a prediction rule: improvement on cross-validation," Journal of the American Statistical Association, 78(382): 316-331.

Farell AE, Roat SD (1994). "Framework for enhancing fault-diagnosis capabilities of artificial neural networks," Computers and Chemical Engineering, 18(7): 613-635.

Flammini A, Marioli D, Taroni A (2001). "A low-cost diagnostic tool for stepping motors," IEEE Transactions on Instrumentation and Measurement, 50(1): 157-162.

Fleming MC, Nellis JG (1994). Principles of Applied Statistics, Routledge.

Foerster J, Lohmann A, Mezger M, RiesMueller K (1997). "Advanced engine misfire detection for SI-engines," SAE Special Publications, vol.1236: 167-173.

Frank PM (1990). "Fault diagnosis in dynamic systems using analytical and knowledge-based redundancy--a survey and some new results," Automatica, 26(3): 459-474.

Frank PM (1994). "On-line fault-detection in uncertain nonlinear-systems using diagnostic observers - a survey," International Journal of Systems Science, 25(12): 2129-2154.

Frank PM, KoppenSeliger B (1997). "New developments using AI in fault diagnosis," Engineering Applications of Artificial Intelligence, 10(1): 3-14.

Frank T, Kraiss KF, Kuhlen T (1998). " Comparative analysis of fuzzy ART and ART-2A network clustering performance," IEEE Transactions on neural networks, 9(3): 544-559.

Fukunaga K (1983). "nonparametric discriminant analysis," IEEE Transactions on Pattern Analysis and Machine Intelligence, 5(6): 671-678.

Fukunaga K (1989). "Estimation of classifier performance," IEEE Transactions on Pattern Analysis and Machine Intelligence, 11(10): 1087-1101.

Fukunaga K (1990). "Introduction to Statistical Pattern Recognition," Academic Press, 2nd Edition.

Fukunaga K, Hayes RR (1989). "Effects of sample size in classifier design," IEEE Transactions on Pattern Analysis and Machine Intelligence, 11(8): 873-885.

Gertler J (1993), "Residual generation in model-based fault diagnosis," Control-theory and advanced technology, 9(1): 259-285.

Gertler J, Costin M, Fang XW (1995). "Model based diagnosis for automotive engines--algorithm development and testing on a production vehicle," IEEE Transactions on Control Systems Technology, 3(1): 61-69.

Gioutsos T (1995). "Signal processing for automotive applications," ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, vol.5: 2975-2978.

Grimmelius HT, Meiler PP, Maas HLMM, Bonnier P, Grevink JS, van Kuilenburg RF (1999). "Three state-of-the-art methods for condition monitoring," IEEE transactions on Industrial Electronics, 46(2): 407-416.

Guglielmi G, Parisini T, Rossi G (1995). "Keynote paper - fault-diagnosis and neural networks - a power- plant application," Control Engineering Practice, 3(5): 601-620.

Guyon I, Makhoul J, Schwartz R, Vapnik V (1998). "What size test set gives good error rate estimates," IEEE Transactions on pattern analysis and machine intelligence, 20(1): 52-63.

Hamamoto Y, Uchimura S, Tomita S (1997). "A bootstrap technique for nearest neighbor classifier design" IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(1): 73-79.

Hara K, Nakayama K (1998). "Training data selection method for generalization by multilayer neural networks," IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences, E81A(3): 374-381.

Haussler D (1992). "Decision Theoretic Generalizations of the PAC Model for Neural Net and Other Learning Applications," Information and Computation, 100(1): 78-150.

Haussler D, Kearns M, Seung HS, Tishby N (1997). "Rigorous Learning Curve Bounds from Statistical Mechanics," Machine Learning, 25: 195-236.

Haykin S (1999). "Neural networks: a comprehensive foundation," Macmillan College Publishing Company, Inc.

Hayton P, Schölkopf B, Tarassenko L, Anuzis P (2001). "Support vector novelty detection applied to jet engine vibration spectra," Advances in Neural Information Processing Systems, vol.13: 946-952

Heinke D, Hamker FH (1998). "Comparing neural networks: A benchmark on growing neural gas, growing cell structures, and fuzzy ARTMAP," IEEE Transactions on Neural Networks, 9(6): 1279-1291

Holmstrom L, Koistinen P, Laaksonen J, Oja E (1997). "Neural and statistical classifiers-taxonomy and two case studies," IEEE Transactions on Neural Networks, 8(1): 5-17.

Hsu CH, Lin CJ (2002). "A comparison of methods for multiclass support vector machines," IEEE Transactions on Neural Networks, 13(2): 415-425.

Hsu PL, Lin KL, Shen LC (1995). "Diagnosis of multiple sensor and actuator failures in automotive engines," IEEE Transactions On Vehicular Technology, 44(4): 779-789.

Huang GB, Chen YQ, Babri HA (2000) "Classification ability of single hidden layer feedforward neural networks," IEEE Transactions on Neural Networks, 11(3): 799-801.

Huang SC, Huang YF (1991)."Bounds on the number of hidden neurons in multilayer perceptrons" IEEE Transactions on Neural Networks, 2(1): 47-55.

Huang WY, Lippmann RP (1988) "Neural net and traditional classifiers," in Neural Information Processing Systems, Anderson D, Ed. New York: American Institute of Physics, 387-396.

Hudon S, Yan Y, Kinsner W (1990). "A comparative-study of neural network models," Mathematical and Computer Modelling, 14: 300-304.

Hurtado JE, Alvarez DA (2001). "Neural-network-based reliability analysis: a comparative study," Computer Methods in Applied Mechanics and Engineering, 191(1-2): 113-132.

Hwang YS, Bang SY (1997). "Recognition of unconstrained handwritten numerals by a radial basis function neural network classifier," Pattern Recognition Letters, 18(7): 657-664.

Intel51. "MCS(R) 51 Microcontroller Family User's Manual," http://www.intel.com/design/mcs51/docs_mcs51.htm#Manuals.

Intel96. "8XC196Kx, 8XC196Jx, 87C196CA Microcontroller Family User's Manual," http://www.intel.com/design/mcs96/docs_mcs96.htm#Manuals.

Isermann R (1984). "Process fault detection based on modelling and estimation methods--a survey," Automatica, 20(4): 387-404.

Isermann R (1993). "Fault diagnosis of machines via parameter estimation and knowledge processing--tutorial paper," Automatica, 29(4): 815-835.

Isermann R (1997). "Supervision, fault-detection and fault-diagnosis methods-an introduction," Control Engineering Practice, 5(5): 639-652.

Isermann R, Balle P (1997). "Trends in the application of model-based fault detection and diagnosis of technical processes," Control Engineering Practice, 5(5): 709-719.

Isermann R, Raab U (1993). "Intelligent actuators - ways to autonomous actuating systems," Automatica, 29(5): 1315-1331.

Jack LB, Nandi AK (2001). "Support vector machines for detection and characterization of rolling element bearing faults," Proceedings of the Institution of Mechanical Engineers Part C-Journal of Mechanical Engineering Science, 215(9): 1065-1074.

Jain AK, Duin RPW, Mao JC (2000). "Statistical pattern recognition: A review," IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(1): 4-37.

Jain AK, Mao JC (1997). "Special issue on artificial neural networks and statistical pattern recognition," IEEE Transaction on neural networks, 8(1): 1-3.

Joshi A, Ramakrishman N, Houstis EN, Rice JR (1997). "On neurobiological, neuro-fuzzy, machine learning, and statistical pattern recognition techniques," IEEE Transactions on Neural Networks, 8 (1): 18-31.

Kavuri SN, Venkatasubramanian V (1993a). "Representing Bounded Fault Classes Using Neural Networks With Ellipsoidal Activation Functions," Computers & Chemical Engineering, 17(2): 139-163.

Kay JW, Titterington DM (1999). Statistics and Neural Networks: Advances at the Interface, Oxford University Press.

Kim YW, Rizzoni G, Samimy B, Wang YY (1995). "Analysis and processing of shaft angular velocity signals in rotating machinery for diagnostic applications," ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, vol.5: 2971-2974.

Kittler J, Hatef M, Duin RPW, Matas J (1998). "On combining classifiers," IEEE Transactions on Pattern Analysis & Machine Intelligence, 20(3): 226-239.

Kobayashi H; Onoye T; Shirakawa I (2002). "Performance estimation at architecture level for embedded systems," IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences, E85A(12): 2636-2644.

Kramer MA, Leonard JA (1990). "Diagnosis Using Backpropagation Neural Networks - Analysis and Criticism," Computers & Chemical Engineering, 14(12): 1323-1338.

Lada EK, Lu JC, Wilson JR (2002). "A wavelet-based procedure for process fault detection," IEEE Transactions on Semiconductor Manufacturing, 15(1): 79-90.

Lampariello F, Sciandrone M (2001) "Efficient training of RBF neural networks for pattern recognition," IEEE Transactions on Neural Networks, 12(5): 1235-1242.

Lampariello F, Sciandrone M (2001). "Efficient training of RBF neural networks for pattern recognition," IEEE Transactions on Neural Networks, 12(5): 1235-1242.

Lawrenz, W (1997). CAN System Engineering, Springer-Verlag.

Le Riche R, Gualandris D, Thomas JJ, Hemez F (2001). "Neural identification of non-linear dynamic structures," Journal of Sound and Vibration, 248(2): 247-265.

Le TT, Watton J, Pham DT (1998). "Fault classification of fluid power systems using a dynamics feature extraction technique and neural networks," Proceedings of the Institution of Mechanical Engineers. Part I, Journal of Systems & Control Engineering, 212(2): 87-97.

Lee C, Landgrebe DA (1997). "Decision boundary feature extraction for neural networks," IEEE Transactions on neural networks, 8(1): 75-83.

Lee S, Kil RM (1991). "A Gaussian potential function network with hierarchically self-organizing learning," Neural networks, 4: 207-224.

Leen G, Heffernan D, Dunne A (1999). "Digital networks in the automotive vehicle," Computing and Control, 10(6): 257-266.

Leonard JA, Kramer MA (1990). "Classifying process behaviour with neural networks: strategies for improved training and generalization," Proceedings of the American Control Conference, San Diego, CA, USA, 23-25 May 1990, (Conf. code 14483), no.1990: 2478-2483.

Leonhardt S, Ayoubi M (1997). "Methods of fault diagnosis," Control Engineering Practice, 5(5): 683-692.

Leung FHF; Lam HK; Ling SH; Tam PKS (2003) "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," IEEE Transactions on Neural Networks, 14(1): 79-88.

Li Q, Tufts DW (1997). "Principal feature classification," IEEE Transactions on neural networks, 8(1): 155-160.

Li YH, Jones NB, Pont MJ (1998). "Applying neural networks and fuzzy logic to fault diagnosis: a review," Proceedings of recent advances in soft computing'98, Leicester, UK: 104-119.

Li Z, Akishita S, Kato T (1997). "Engine failure diagnosis with sound signal using wavelet transform," SAE Special Publications, vol.1240: 79-86.

Li Z, Akishita S, Tokumoto SI, Kato T (1996). "Failure diagnosis system by sound signal for automobile engine," Proceedings of the Japan/USA Symposium on Flexible Automation, vol.1: 427-430.

Lippmann RP (1987). "An introduction to computing with neural networks," Institute of Electrical and Electronic Engineers (USA): Acoustics, Speech and Signal Processing.

Lippmann RP (1989). "Pattern classification using neural networks," IEEE Communications Magazine, 27(11): 47-64.

Lippmann RP, in Cherkassky V, Friedman JH, Wechsler H (Eds) (1994). "From statistics to neural networks : theory and pattern recognition applications," Berlin; London : Springer-Verlag.

Liu JH; Gader P (2002). "Neural networks with enhanced outlier rejection ability for off-line handwritten word recognition," Pattern Recognition, 35(10): 2061-2071.

Liu S, Gu F, Ball A (2002). "The on-line detection of engine misfire at low speed using multiple feature fusion with fuzzy pattern recognition," Proceedings of The Institution of Mechanical Engineers Part D-Journal of Automobile Engineering, 216 (5): 391-402.

Looney CG (1997). Pattern Recognition Using Neural Networks. Oxford University Press.

MacGregor R, "OBD II Explained," http://www.vru.com/carshow/obd2.html.

Mak MW, Allen WG, Sexton GG (1993). "Comparing multi-layer perceptrons and radial basis function networks in speaker recognition," Journal of Microcomputer Applications, 16(2): 147-159.

Mak MW, Allen WG, Sexton GG (1994). "Speaker identification using multilayer perceptron and radial basis function networks," Neurocomputing, 6(1): 99-117.

Maki Y, Loparo KA (1997). "A neural-network approach to fault detection and diagnosis in industrial processes," IEEE Transactions on Control Systems Technology, 5(6): 529-541.

Mangasarian OL, Wolberg WH (1990). "Cancer diagnosis via linear programming," SIAM News, 23(5): 1-18.

Manson G, Worden K, Holford K, Pullin R (2001). "Visualisation and dimension reduction of acoustic emission data for damage detection," Journal of Intelligent Material Systems and Structures, 12(8): 529-536.

Marks RJ (1993). "Intelligence: computational versus artificial," IEEE Transactions on Neural Networks, 4(5): 737-739.

Martinez D (1998). "Neural tree density estimation for novelty detection," IEEE Transactions on Neural Networks, 9(2): 330-338.

Marzi H (2002). "Development of a real-time monitoring system," Proceedings of the Institution of Mechanical Engineers Part B-Journal of Engineering Manufacture, 216(6): 933-937.

Mehrotra KG, Mohan CK, Ranka S (1991). "Bounds on the number of samples needed for neural learning" IEEE Transactions on Neural Networks, 2(6): 548-558.

Michie D, Spiegelhalter DJ, Taylor CC (Eds) (1994). "Machine learning, neural and statistical classification," New York; London: Ellis Horwood.

Microsoft (1992). MASM, Reference, Assembly-Language Development System Version 6.1, Document No. DB35749-1292.

Min BK, O'Neal G, Koren Y, Pasek Z (2002). "A smart boring tool for process control," Mechatronics 12(9-10): 1097-1114.

Moody J, Darken CJ (1988). "Learning with localised receptive fields," Proceedings of the 1988 connectionist models summer school, 133-143.

Moody J, Darken CJ (1989). "Fast learning in networks of locally-tuned processing units," Neural computation, 1(2): 281-294.

Moro D, Azzoni P, Minelli G (1998). "Misfire pattern recognition in high performance SI 12 cylinder engine," SAE Special Publications, vol.1357: 87-94.

Musavi MT, Chan KH, Hummels DM, Kalantri K (1994). "On the generalization ability of neural network classifiers," IEEE Transactions on Pattern Analysis and Machine Intelligence, 16(6): 659-663.

Neal RM (1996). Bayesian Learning for Neural Networks, Springer-Verlag, ISBN: 0-387-94724-8.

Newland DE (1993). "An introduction to random vibrations, spectral and wavelet analysis," Harlow : Longman Scientific & Technical.

Newland DE (1994). "Wavelet analysis of vibration .1. Theory; 2. Wavelet maps," Journal of Vibration and Acoustics-Transactions of the ASME, 116(4): 409-425.

Oppenheim AV (1975). "Digital signal processing" London : Prentice Hall.

Orr MJL (1999). "Recent Advances in Radial Basis Function Networks," Tech. Rep., Center for Cognitive Science, University of Edinburgh.

Ozyurt B, Kandel A (1996). "A hybrid hierarchical neural network-fuzzy expert-system approach to chemical process fault-diagnosis," Fuzzy sets and systems, 83(1): 11-25.

Parikh CR, Pont MJ, Li Y, Jones NB (2000). "Investigating the performance of MLP classifiers where limited training data are available for some classes," in R. John and R. Birkenhead (Eds), Advances in Soft Computing--Soft Computing Techniques and Applications, Springer-Verlag, ISBN 3-7908-1257-9: 22-27.

Parikh CR, Pont MJ, Li YH, Jones NB (1999). "Neural networks for condition monitoring and fault diagnosis: the effect of training data on classifier performance," Proceedings of the International conference on condition monitoring, Swansea, UK: 237-244.

Patton R, Frank PM, Clark R (1989). "Fault diagnosis in dynamic systems-theory and applications," Prentice Hall.

Patton RJ, Chen J (1997). "Observer-based fault detection and isolation: Robustness and applications," Control Engineering Practice, 5(5): 671-682.

Patton RJ, Chen J, Nielsen SB (1995). "Model-Based Methods for Fault-Diagnosis - Some Guide Lines," Transactions of the Institute of Measurement and Control, 17(2): 73-83.

Patton RJ, Lopez-Toribio CJ (1998). "Artificial intelligence approaches to fault diagnosis," IEE Colloquium, Update on Developments in intelligent Control, 23 Oct 1998.

Petrilli O, Paya B, Esat II, Badi MNM (1995). "Neural networks based fault detection using different signal processing techniques as pre-processing," American Society of Mechanical Engineers, Petroleum Division (Publication) PD, 70: 97-101.

Pittner S, Kamarthi SV (1999). "Feature extraction from wavelet coefficients for pattern recognition tasks," IEEE Transactions on Pattern Analysis and Machine Intelligence, 21(1): 83-88.

Plutowski M, White H (1993). "Selecting concise training sets from clean data," IEEE Transactions on Neural Networks, 4(2): 305-318.

Pont MJ (2001). Patterns for Time-Triggered Embedded Systems, Addison-Wesley Publishers Ltd.

Powell MJD (1987). " Radial basis functions for multivariable interpolation: a review," Conference in Algorithms for the Approximation of Functions and Data.

Prakash KN, Ramakrishnan AG, Suresh S, Chow TWP (2002). "Fetal lung maturity analysis using ultrasound image features," IEEE Transactions on Information Technology in Biomedicine, 6(1): 38-45.

Prechelt L (1994). "PROBEN1-A set of benchmarks and benchmarking rules for neural network training algorithns," Tech. Report 21/94, Fakultat fur

Informatik, Universitat Karlsruhe, Germany, FTP: ftp.ira.uka.de/pub/papers/techreports/1994/1994-21.ps.Z

Prechelt L (1996). "A Quantitative Study of Experimental Evaluations of Neural Network Learning Algorithms: Current Research Practice," Neural Networks, 9(3): 457-462

Rao BKN (ed.) (1996). Handbook of Condition Monitoring, Oxford: Elsevier Advanced Technology.

Rao NSV (1999) "Simple sample bound for feedforward sigmoid networks with bounded weights," Neurocomputing, 29 (1-3): 115-122.

Rathbun TF, Rogers SK, DeSimio MP, Oxley ME (1997). "MLP iterative construction algorithm," Neurocomputing, 17 (3-4): 195-216.

Ribbens WB, Bieser S (1995). "Advanced signal processing for misfire detection in automotive engines," ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, vol.5: 2963-2966.

Ribbens WB, Park J, Kim D (1994). "Application of neural networks to detecting misfire in automotive engines," ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, vol.2: 593-596.

Ripley BD (1994). "Neural networks and related methods for classification (with discussion)," Journal of the Royal Statistical Society Series B-Methodological, 56(3): 409-456.

Ripley BD (1995). "Statistical ideas for selecting network architectures," In 'Neural Networks: Artificial Intelligence and Industrial Applications,' eds B. Kappen & S. Gielen, Springer, 183-190.

Roberts SJ (1999). "Novelty detection using extreme value statistics," IEE Proceedings-Vision Image and Signal Processing, 146(3): 124-129.

Ross TJ (1995). "Fuzzy logic with engineering applications," McGraw-Hill, Inc.

Rozier D (2001). "A strategy for diagnosing complex multiple-fault situations with a higher accuracy/cost ratio," Engineering Applications of Artificial Intelligence, 14(2): 217-227.

Rumelhart D, McClelland JL (1986). Parallel Distributed Processing, vol.1. M.I.T. Press.

Sanchez VD (1996). "On the design a class of neural networks," Journal of network and computer applications, 19: 111-118.

Scholkopf B, Sung KK, Burges CJC, Girosi F, Niyogi P, Poggio T, Vapnik V (1997). "Comparing support vector machines with Gaussian kernels to radial basis function classifiers," IEEE Transactions on Signal Processing, 45(11): 2758-2765.

Schwenker F, Kestler HA, Palm G (2001). "Three learning phases for radial-basis-function networks," Neural Networks 14(4-5): 439-458.

Setiono R (2001) "Feedforward neural network construction using cross validation," Neural Computation, 13(12), 2865-2877.

Sick B (2002). "On-line and indirect tool wear monitoring in turning with artificial neural networks: A review of more than a decade of research," Mechanical Systems and Signal Processing, 16(4): 487-546.

Silipo R, Marchesi C (1998). "Artificial neural networks for automatic ECG analysis," IEEE Transactions on Signal Processing, 46(5): 1417-1425.

Skoundrianos EN, Tzafestas SG (2002). "Fault diagnosis via local neural networks," Mathematics and Computers in Simulation, 60 (3-5): 169-180.

Skurichina M, Duin RPW (1996). "Stabilizing classifiers for very small sample sizes," ICPR13, Proc. 13th Int. Conf. on Pattern Recognition (Vienna, Austria, Aug.25-29) vol.2, Track B: Pattern Recognition and Signal Analysis, IEEE Computer Society Press, Los Alamitos: 891-896.

Smolensky P, Mozer MC, Rumelhart DE (1996). Mathematical Perspectives on Neural Networks, Mahwal, New Jersey, Lawrence Erlbaum Associates, Publishers.

Smyth P (1994). "Markov monitoring with unknown states," IEEE Journal on Selected Areas in Communications, 12(9): 1600-1612.

Somol P, Pudil P (2002). "Feature selection toolbox," Pattern Recognition, 35: 2749-2759.

Staszewski WJ (1998). " Wavelet based compression and feature selection for vibration analysis," Journal of Sound and Vibration, 211(5): 735-760.

Staszewski, WJ (2000). "Advanced data pre-processing for damage identification based on pattern recognition," International Journal of Systems Science, 31(11): 1381-1396.

Staszewski, WJ, Worden K (1997). "Classification of faults in gearbox – pre-processing algorithms and neural networks," Neural Computing & Applications, 5(3): 160-183.

Stitson MO, Weston JAE, Gammerman A, Vovk V, Vapnik V (1996). "Theory of support vector machines," Technical Report CSD-TR-96-17, Department of Computer Science, Royal Holloway, University of London.

Takahashi H, Gu HZ (1998). "A tight bound on concept learning," IEEE Transactions on neural networks, 9(6): 1191-1202.

Tandon N, Choudhury A (1999). "A review of vibration and acoustic measurement methods for the detection of defects in rolling element bearings," Tribology International, 32(8): 469-480.

Tarassenko L, Nairac A, Townsend N, Buxton I, Cowley Z (2000). "Novelty detection for the identification of abnormalities," International Journal of Systems Science, 31(11): 1427-1439.

Tarassenko L, Nairac A, Townsend NW, Buxton I, Cowley P (2000). "Novelty detection for the identification of abnormalities" Int J Systems Science, 11: 1427-1439

Tarassenko L, Roberts S (1994). "Supervised and unsupervised learning in radial basis function classifiers," IEE Proceedings-Vision Image and Signal Processing, 141(4): 210-216.

Terra MH, Tinos R (2001). "Fault detection and isolation in robotic manipulators via neural networks: A comparison among three architectures for residual analysis," Journal of Robotic Systems, 18 (7): 357-374.

Theodoridis S, Koutroumbas K (1999). Pattern Recognition. Academic Press.

Tsoi AC, Back A (1995). "Static and dynamic preprocessing methods in neural networks," Engineering application of artificial intelligent, 8(6): 633-642.

Twiddle JA (1999). Fuzzy Model Based Fault Diagnosis of a Diesel Engine Cooling System. Department of Engineering, University of Leicester, Report 99-1 Jan 1999.

Vahid F, Givargis T (2002). Embedded System Design: a Unified Hardware/Software Introduction. New York: John Wiley & Sons, Inc.

Wang CC, Too GPJ (2002). "Rotating machine fault detection based on HOS and artificial neural networks," Journal of Intelligent Manufacturing, 13(4): 283-293.

Watanabe K, Hirota S, Hou L, Himmelblau DM (1994). "Diagnosis of Multiple Simultaneous Fault Via Hierarchical Artificial Neural Networks," AICHE Journal, 40(5): 839-848.

Watkins DS (1991). "Fundamentals of Matrix Computations," John Willey & Sons, Inc.

Williams J (1996). "An overview of misfiring cylinder engine diagnostic techniques based on crankshaft angular velocity measurements," SAE special publications, 1149: 31-37.

Williams J (1996a). "Improved methods for digital measurement of torsional vibration," Proceedings of the 1996 international truck & bus meeting & exposition, conference code 45559, Detroit MI, USA: 9-15.

Willsky AS (1976). "A survey of design methods for failure detection in dynamic systems," Automatica, 12(6): 601-611.

Wilmshurst T (2001). An Introduction to the Design of Small-Scale Embedded Systems. Basingstoke: Palgrave, 2001.

Wilson CL; Blue JL; Omidvar OM (1997). "Training dynamics and neural network performance," Neural Networks, 10(5): 907-923.

Wu ZJ, Lee A (1998). "Misfire detection using a dynamic neural network with output feedback," SAE Special Publications, vol.1357: 33-37.

Yang DM, Stronach AF, MacConnell P, Penman J (2002). "Third-order spectral techniques for the diagnosis of motor bearing condition using artificial neural networks," Mechanical Systems and Signal Processing, 16(2-3): 391-411.

Ye N, Zhao B (1996). "A hybrid intelligent system for fault-diagnosis of advanced manufacturing system," International journal of production research, 34(2): 555-576

Ypma A., Duin RPW (1997). "Novelty detection using self-organizing maps," in: N. Kasabov, R. Kozma, K. Ko, R. O'Shea, G. Goghill, T. Gedeon (eds.), Progress in Connectionist-Based Information Systems, vol.II, Springer Verlag, Berlin, 1322-1325.

Yu D, Shields DN, Daley S (1996). "A hybrid fault-diagnosis approach using neural networks," Neural computing & applications, 4(1): 21-26.

Zadeh LA (1965). "Fuzzy sets," Information and Control, 8: 338-353.

Zadeh LA (1973). "Outline of a new approach to the analysis of complex systems and decision processes," IEEE Transactions on System, Man, and Cybernetics, 3(1): 28-44.

Zhang GQP (2000). "Neural networks for classification: a survey," IEEE Transactions on Systems Man and Cybernetics Part C - Applications and Reviews, 30(4): 451-462.

Zheng ZJ (1993). "A benchmark for classifier learning," Tech. Rep. TR474, Basser Department of Computer Science, University of Sidney, Anonymous ftp:/ftp.cs.su.oz.au/pub/tr/TR93_474.ps.Z.