SYMMETRIC RADIAL BASIS FUNCTION NETWORKS AND THEIR APPLICATION TO VIDEO DE-INTERLACING

Alfredo Giani

A thesis submitted for the degree of

Doctor of Philosophy

University of Southampton

Faculty of Engineering and Applied Science

Institute of Sound and Vibration Research

December 2002

University of Southampton

Abstract

Faculty of Engineering and Applied Science
Institute of Sound and Vibration Research

Doctor of Philosophy

Symmetric Radial Basis Function Networks and their Application to Video De-interlacing

by Alfredo Giani

Video technology is one of the most advanced fields of modern electronics. The degree of interaction between different disciplines is simply extraordinary. In particular, the introduction of digital signal processing has revolutionised video technology. Sophisticated mathematical techniques are applied at several points in the video chain. One of the most common tasks in video processing is de-interlacing, i.e. the conversion from an interlaced format to a progressive format. De-interlacing can be considered as a form of interpolation, where the even lines of a frame in a video are estimated by interpolating the odd lines in the frame. De-interlacing is a task that is often required in real time, therefore the amount of computation required is a critical parameter. A straightforward way of achieving this task is by employing some form of linear interpolation. Linear techniques are fast, robust, and theoretically tractable. Unfortunately, the results are often unsatisfactory in terms of image quality. A more sophisticated approach is offered by the implementation of non-linear techniques. When dealing with non-linear methods, two main factors must be considered: the amount of computation required and the tendency to over-fit the data (generalisation). In this work two non-linear techniques are investigated, the Volterra series and the Radial Basis Function Networks (RBFN). Volterra series can be considered as an extension of the linear model to a polynomial series. RBFN is a functional series that creates arbitrary interpolation mappings due to its localisation properties. It is shown how both techniques produce results that are superior compared to linear techniques in terms of image quality. However great care must be taken and proper training procedures must be devised in order to maintain the computational load to an acceptable level and to avoid over-fitting. In this thesis a novel approach is devised to reduce the computational load in RBFN at the same time increasing the generalisation ability of the network by exploiting the symmetries arising in the input space when the sampling lattice is symmetric. Symmetrisation is initially developed for linear techniques, where an analytic derivation leads to a reduction in the number of operations required. Successively this technique is extended to RBFN by devising a mathematical approach that is both intuitive and rigorous. It is shown how symmetrisation leads to a reduction of the computational load required at the same time increasing the generalisation ability of the interpolation.

TABLE OF CONTENTS

Appendices and bibliography

PREFACE

Introduction

Video technology is one of the most advanced fields of modern electronics. The degree of interaction between different disciplines is simply extraordinary. Optics, analog and digital electronics, analog and digital signal processing, psychology of visual perception, all contribute to create the field. In particular, the introduction of digital signal processing has revolutionised video technology. Sophisticated mathematical techniques are applied at several points in the video chain, from the camera that creates the video signal to the device that displays it.

One of the points that engineers have to tackle is the necessity of communication and interaction between devices that work to different standards. This has always been an issue since the early days of the entertainment industry. On one side is the desire to make video production available to the broadest audience that pushes for the integration of different systems and for a complete update of the available technology whenever new developments are ready for the market. On the other side, it is commercially unthinkable to force customers to completely renew their home systems; the history of consumer technological products is full of blunders, that failed because customers were just not ready to spend money on them. Since the diversity of standards arose at the beginning (e.g., the American-Japanese NTSC standard, against the European PAL), the result is that a plethora of different systems are forced to interact. This diversity has not changed with the advent of digital technology and high definition television (HDTV). On the contrary, a new challenge has been posed, since it is now necessary to convert the immense video production available from the old standards to the new one, and vice-versa. In the effort to convert a video signal from one standard to another, it is vital to preserve the quality of the video images as much as possible.

In this struggle to make different worlds live together, the *de-interlacing of video images* plays a remarkable role. In simple words, de-interlacing is a form of interpolation, as we shall see shortly. De-interlacing is required whenever it is

necessary to modify the rate at which images are presented (frame rate), the dimension of the image, etc. Conventional interpolation is performed using linear techniques. Linear interpolation is fast, easy to implement in both software and hardware, and has well-assessed theoretical background. Unfortunately, its application is limited by the fact that de-interlacing is basically a non-linear problem. The result is that the performance of linear de-interlacers is often unsatisfactory. Non-linear interpolation techniques, noticeably the polynomial series, have been recently studied, and they show remarkable improvements compared to the linear approach. Unfortunately the amount of computation required is a major limitation to the application of non-linear techniques.

The main subject of this work is the study of a particular form of non-linear interpolation scheme, called the *Radial Basis Function Network* (RBFN). The key points of this technique are its scalability, its high degree of architectural parallelism, and its connection with many aspects of signal processing that together make RBFN a very flexible and robust interpolation technique. The task of this work is to find a reasonable way to build an RBFN-based interpolation scheme with a quality comparable to polynomial series, and with comparable, if not reduced, computational burden.

In pursuing this task, we have developed a novel technique that exploits some peculiar properties of image processing to reduce the computational complexity of RBFN interpolators. This approach is based on the symmetry properties of the sampling lattices used to perform image interpolation. These properties are successfully applied to simplify the computational demands of linear interpolation. We attempted to transfer these properties to the RBFN model. This task is performed considering the topological properties of RBFN in the input space. Although the results can not be considered definitive, the path is promising. We reserve future efforts to complete the framework depicted in this work.

Description of the chapters

In this section, we give a concise description of contents of each of the chapters that compose this work. The chapters have been grouped into two parts. The first part

(chapter 1 to 7) is focused on the description of the de-interlacing problem, and of the theory and application of the currently available technology, specifically linear interpolation, Volterra series, and RBFN.

The second part (chapter 8 to 10) contains the majority of the original contributions of this work, the analysis of the theoretical and practical consequences of the symmetry constraints in image interpolation, and the consequent development of different techniques that are able to reduce the computational demands of RBFN.

Part 1. RBFN de-interlacing

Chapter 1. Introduction to de-interlacing

In this chapter, we give a general view of the de-interlacing problem. We introduce the concepts of interlaced and progressive video, and the reasons why the first technology was chosen in the early days of video technology. De-interlacing is then modelled as an interpolation problem, and we describe the problems arising when we perform this interpolation using standard linear methods. These problems justify the use of non-linear techniques.

Chapter 2. Single-layer networks

This chapter introduces the general concepts of single-layer networks, and many of the issues associated with this model. The single-layer network is a general model that includes linear interpolation, Volterra series, and RBFNs. As such, this chapter is a necessary introduction to the study and application of these techniques to the interpolation problem. Some general concepts of system modelling and pattern learning are introduced. The concepts of parametric models, learning by examples, and cost functions are presented.

Chapter 3. Radial Basis Function Networks

This chapter gives a thorough description of the RBFN model. A panoramic view of the many theoretical aspects of RBFNs is given, and the most popular RBFN schemes are introduced. The chapter shows the links between RBFN and two main subjects of

research, the theory of probabilistic neural networks, and the theory of functional approximation.

Chapter 4. Orthogonal Least Squares Algorithm

This chapter describes the Orthogonal Least Squares Learning algorithm (OLS). It provides an overview of the basic algorithm, together with some of the theoretical aspects of orthogonal least squares techniques that motivated the development of OLS. The practical algorithms applied in the successive sections are also fully described.

Chapter 5. Wiener and Volterra de-interlacing

This chapter investigates the practical application of linear filters and the Volterra polynomial series to de-interlace a sample frame. Linear interpolation is the technique currently used in most of the broadcast systems, and previous work has considered the application of Volterra series. An initial cost analysis on the Volterra series is performed, motivating the choice of a particular sampling structure that trades off performance and cost. OLS is applied to the Volterra series, and it is shown how the different kernels are relevant to the performance according to their order.

Chapter 6. RBFN de-interlacing

In this chapter, a series of experiments are used to explore the advantages and limitations of RBFN to de-interlace a frame. Initially, a simple training strategy is devised to avoid the problems associated with the training of non-linear parameters in RBFN. OLS is applied throughout the chapter as the main technique to achieve reduced complexity at the same time striving to maintain performance. Successively, more sophisticated learning procedures are proposed and applied. A comparison with Volterra series is conducted. This comparison is based on the simple cost analysis described in appendix B.

Chapter 7. Generalisation

One of the main issues in pattern recognition is the ability to produce systems that, trained on a limited set of data, are able to produce the same performance on a much

larger set of inputs. This ability is called generalisation. A realistic de-interlacing system must yield satisfactory results on a very large set of inputs and it is clear that a learning procedure must deal with a much smaller subset of training patterns.

This chapter firstly presents an overview of many of the theoretical aspects involved in generalisation. This field of research has a large theoretical background, and it is obviously impossible to present all the aspects of generalisation theory and all the techniques available in this work. Therefore, the relevant theoretical aspects of generalisation will be introduced, together with few practical techniques to achieve general training. A series of experiments will be applied using a set of six frames, and the system is evaluated whereas training will only employ single frames.

Part 2. Symmetric RBFN de-interlacing

Chapter 8. Exploiting symmetry for linear filters

This chapter introduces the theoretical consequences of the symmetric sampling of images. In many interpolation schemes, it is likely that the sampling lattice has a number of symmetries with respect to the target pixel. The symmetric properties of the sampling lattice and some general consideration of images will lead to a set of conditions that an image interpolation system should abide by. A simple analytical derivation based on these considerations leads to the conclusion that it is possible to reduce the computational burden required by linear de-interlacing.

Chapter 9. Exploiting symmetry in Radial Basis Function Networks

This chapter describes the attempt to apply the properties of symmetry to a RBFN interpolation scheme. Initially, the problems of extending the linear approach to the non-linear case are introduced. Two interesting theoretical developments that lead to two reduced RBFN architectures, the symmetric RBFN model and the folded RBFN, are studied and applied. It is shown how it is possible to achieve computational savings using these two techniques.

Chapter 10. Generalisation of symmetric strategies

Symmetric structures are based on the idea that an interpolation system must yield the same performance over a rotated version of the input image. By doing so, somehow the system increases its generality, since it produces similar performance on different versions of a given image. This chapter demonstrates that this is the case, specifically that by reducing the computational load using symmetric techniques, one also improves the generality of the system, providing further reason to implement symmetric training strategies.

Chapter 11. Conclusions

This section summarises and discusses the results obtained in the previous sections, providing hints for future developments.

Appendices

Appendix A. Fourier spectra of the video signal

This appendix is a brief study of Fourier analysis applied to video sequences. In particular it explains the theoretical background to understand why de-interlacing is fundamentally a non-linear problem.

Appendix B. Cost considerations

In this appendix a simple cost comparison between the techniques applied in this work is developed, in order to compare their relative cost in terms of computational requirements. This cost comparison is based on reasonable considerations of the relative cost of the elementary computational units involved. A more accurate cost analysis should require the study of hardware and software issues that are out of the scope of this thesis.

Appendix C. Nelder-Mead simplex algorithm

This appendix briefly describes a popular iterative optimisation technique, that is applied to the non-linear parameters of RBFN to produce further improvements to the performance of RBFN de-interlacers.

Acknowledgements

Finally, I wish to express my considerable gratitude to everybody who, in a way or another, have helped me in the process of writing this thesis. Firstly, I wish to thank my parents. They deserve my gratitude if only for the simple fact that they gave me birth. Unlimited gratitude goes to Donatella. Her support during these years has been priceless, and she gave me motivation to pursue a career in research.

I wish to thank all the people, students and academics that I had the pleasure to meet in the ISVR. Of course, my gratitude goes to my supervisor, mentor and friend Dr. Paul White. His unique mixture of competence, modesty, humanity and great sense of humour has been one of the best lessons I ever got from life, and greatly contributed to make me enjoy the pursue of this work.

I apologise to anyone who thinks he/she should have an explicit place in this section, and that I did not mention for evident reasons of space. I can only say that readers usually do not pay much attention to the acknowledgements, so it is not the case to break a good friendship for that.

Nomenclature

The symbols used for the most commonly occurring quantities in the thesis are listed below:

n	training pattern label
L	number of patterns
j	input label
D	dimension of the input space
M	number of nodes in the network
p	degree of the Volterra series
x	input variable
$\mathbf{x}$	input vector variable
$\mathbf{X}$	input space
h	node output (layer output) variable
$\mathbf{h}$	intermediate vector (layer output) variable
$\mathbf{H}$	intermediate space
t	target output variable
$\mathbf{T}$	target space, symmetry operator[1]
y	network output variable
$\mathbf{Y}$	output space
ϕ	node's weight (linear parameter)
$\mathbf{\Phi}$	node weight (linear parameter)
$\mathbf{c}$	node centre
σ	node width
C	cost function
Ω	regularisation function
ν	regularisation factor

[1] Whenever **T** indicates a symmetry operator (chapters 8, 9 and 10), it is usually shown with a subscript label indicating the angle of symmetry. It is evident from the context whether **T** indicates the sequence of targets (target space) or the generic symmetry operator.

1. INTRODUCTION TO DE-INTERLACING

1.1. Introduction

The creation of continuously varying images, known as video, is a science that has been studied since the XIXth century. Amateur scientists created different mechanical means to entertain the spectators. It was immediately clear then that a sequence of still images, accurately drawn and shown in sequence at an appropriate repetition rate, can create the illusion of motion. The key to this approach is that the human eye acts as a temporal low-pass system, blending still images into a single, continuous flow by means of the phenomenon of image persistence on the retina.

Early studies showed that a *frame rate* (the rate at which the pictures are presented to the eye) of roughly 10 pictures per second is the lower limit needed to create the impression of continuity. Early movie pictures used a frame rate of 24 frames/second. This rate was later doubled to 48 frames/second to reduce the effect of *flickering*. This phenomenon can be described as a rapid change in the luminosity of the picture, due to the decay of retinal persistence between two successive frames. The up-conversion of the frame-rate was obtained by showing the same frame twice, and doubling the shutter speed.

The frame rate partly defines the *scanning format* of the motion picture. Vertical resolution (i.e. the number of horizontal lines in the picture) and horizontal resolution together contribute to define the scanning format. In movie pictures horizontal and vertical resolution are determined by the size of light-sensitive particles in the photographic emulsion on the film. The drawback of having smaller particles (i.e. higher resolution) is that they require more light to be properly developed, reducing the possibility of shooting low-light scenes (Stroebel, Zakia 1993). In television systems vertical and horizontal resolution are critical since they determine the bandwidth requirements of the signal chain.

1.2. Historical background of interlaced video

When engineers attempted to build commercial TV systems in the mid-thirties of the XXth century, they faced a formidable challenge posed by the limited bandwidth available. The frame rate was initially set to the frequency of the power supply line, in order to avoid interference. In the USA and other countries the scanning format was set to 60 frames/second and 525 scanning lines (vertical resolution). In Europe the format was set to 50 frames/second and 625 lines. The horizontal resolution was mainly determined by the available scanning technology (cathodic ray tube, CRT) and by other bandwidth requirements.

Unfortunately, such high frame-rates proved to be uneconomical given the bandwidth requirements, pulling the required base-band up to 10MHz. Using a technique called *interlacing*, engineers managed to compress the base-band of the video signal to the range 0~5 MHz, a requirement that could be fulfilled at the time.

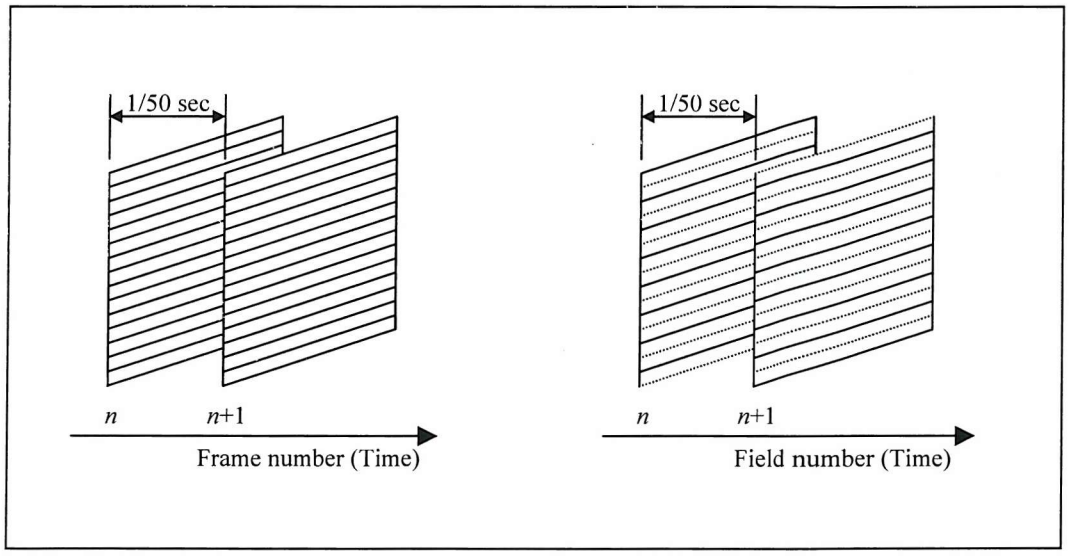

Figure 1. Progressive (left) and interlaced (right) video sequence. Field lines (—), missing lines (--).

Interlacing compresses the bandwidth by alternately showing an image's odd lines (referred to as the odd field) and even lines (even field) as illustrated in figure 1. The rate at which the fields are shown is called the field rate. Given its low-pass characteristic, the human eye in fact integrates the partial information contained in the interlaced picture to recover the missing information. Hence, the viewer observes an

effective frame rate of 50 frames/second (in fact, the field rate) although the real frame rate is 25 frames/second. Unfortunately, this compression comes with the cost of a reduced perceived quality, and with a series of characteristic artefacts.

Interlacing is performed at the camera level, so the missing lines in each field are never measured. Mathematically, interlacing is a vertical sub-sampling of the original picture, and as such, an amount of aliasing is expected (see section 1.4). Unfortunately, this aliasing occurs in the optical path, at the very beginning of the video process, and efficient optical anti-alias filters are still not an economical solution. One example of artefacts introduced by interlacing can be seen by considering a completely dark picture with a single thin horizontal white line. This line may appear in only one of the fields, and its frame rate is in fact halved, and the resulting video will show the line blinking. This effect is called *line flicker*. Another problem arises with moving objects in the picture. If the human eye tracks an object moving in the vertical direction at the speed of one scanning line per field, the effect is that the underlying line structure will be "visible", in the sense that the scan of the horizontal line will be perceivable (Bellers, 1999). This artefact is called *line crawl*, since the perceived impression is that of the scanning line moving at the same speed of the object.

Interlacing preserves vertical resolution, reducing the flicker while yielding an affordable field rate. The downside is that artefacts are introduced, due to high spatial frequencies and motion. These effects however were less obvious at the time interlacing was developed, mainly due to the slow response of the CRT at that time. Hence interlaced video was the choice adopted by broadcast companies.

1.3. De-interlacing

Nowadays, the situation is rather different. The advent of broad-band digital video transmission, the improvement in CRT technology and the development of new, fast video capture and rendering devices like charge coupling devices (CCD) and liquid crystal displays (LCD), makes the transmission of high quality, high frame-rate *progressive* (i.e. non-interlaced) video possible. However, the greatest proportion of video production is currently in interlaced format. Moreover, interlacing is an

effective compression technique, and is required in many *standards-conversion* problems. For these reasons, there is a general interest in studying feasible ways to convert an interlaced signal into progressive format. This task is called *de-interlacing.*

From the signal processing point of view, de-interlacing is a form of interpolation. The task is to estimate the missing lines that, in fact, have never been shot. Hence, in general there is no reference signal to compare results with. Such a reference signal can be estimated using *motion estimation* techniques to move the even-field lines in time to fill the line gaps in the odd field. However, note that such motion estimation is part of the problem. If we are able to produce a "perfect" back-dating at time i of the i+1 field, the problem of de-interlacing field i is trivially solved. In general, a proper de-interlacing scheme requires the interpolation to be performed either in space (*intra-frame* de-interlacing) and in time (*inter-frame* de-interlacing) (Figure 2). An exhaustive treatment of the matter is given in Bellers (1999) and Tekalp (1995).

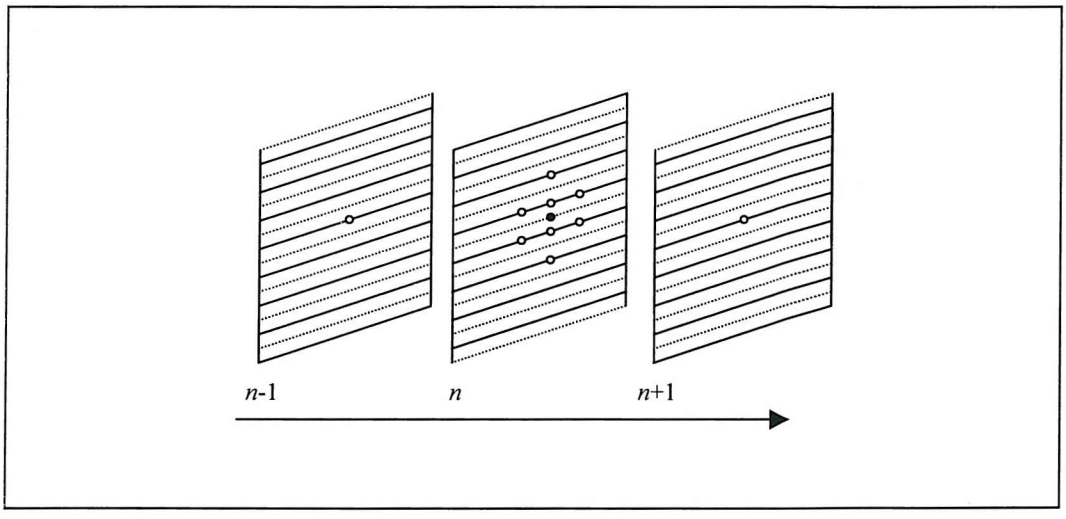

Figure 2 space-time sampling of interlaced video sequences. Intra-field and inter-field pixels are considered (white dots) to estimate the pixel in the missing line of field *n*. (black dot).

In absence of significant motion, inter-frame interpolation is sufficient to estimate the missing lines. However when motion is present, inter-frame interpolation is more complex and requires the use of motion estimators, that in turn require the use of spatial de-interlacers (in order to compute the motion occurring in two successive fields, the missing lines should be estimated). Therefore one can regard spatial de-interlacing as the more low level operation. In this work we will limit our attention to intra-frame (spatial) interpolation. Examples of motion-compensated schemes can be found in Sugiyama, Nakamura(1999), and Jung *et al.* (2000).

In the next section we will discuss the issue of aliasing in video signals. This discussion requires the analysis of the signal in the frequency domain through the Fourier transform.

1.4. Fourier transform of video signals and aliasing

In this section we will discuss the spectral analysis of video signals, both in progressive and interlaced formats. A *continuous* video signal can be modelled as a function $F_c(x,y,t)$ of two continuous spatial variables x, y and the continuous time variable t. Consequently, its complete Fourier transform is a three-dimensional function of the two spatial frequency variables f_x and f_y, and of the temporal frequency f_t

$$\Phi_c(f_x,f_y,f_t)= \iiint F_c(x,y,t)\cdot exp\left[-2\pi\cdot i\left(f_x\cdot x+f_y\cdot y+f_t\cdot t\right)\right]dx\cdot dy\cdot dt \tag{1.1}$$

Clearly, any realistic video-processing system deals with a discretised version of the continuous signal $F_c(x,y,t)$. Typically the sampling occurs on a periodic lattice that defines the scanning format

$$F_d(n,m,k)=F_c(n\cdot\Delta x,m\cdot\Delta y,k\cdot\Delta t) \tag{1.2}$$

where Δx, Δy and Δt are respectively the horizontal, vertical and temporal resolution of the discrete signal. Note that Δt^{-1} is the frame rate of the video sequence. We can write equation (1.2) as a sampling of $F_c(x,y,t)$

$$F_d(n,m,k)=\sum_n\sum_m\sum_k F_c(x,y,t)\delta(x-n\cdot\Delta x)\delta(y-m\cdot\Delta y)\delta(t-k\cdot\Delta t) \tag{1.3}$$

The Fourier transform of $F_d(n,m,k)$ can be expressed as:

$$\Phi_d(f_x,f_y,f_t)=\sum_k\sum_r\sum_s\Phi_c\left(f_x-\frac{q}{\Delta x},f_y-\frac{r}{\Delta y},f_t-\frac{s}{\Delta t}\right) \tag{1.4}$$

that will result in a periodic spectrum as shown in figure 3. The grey circle represents the continuous spectrum (supposed band-limited), and the white circles represent its periodic replications.

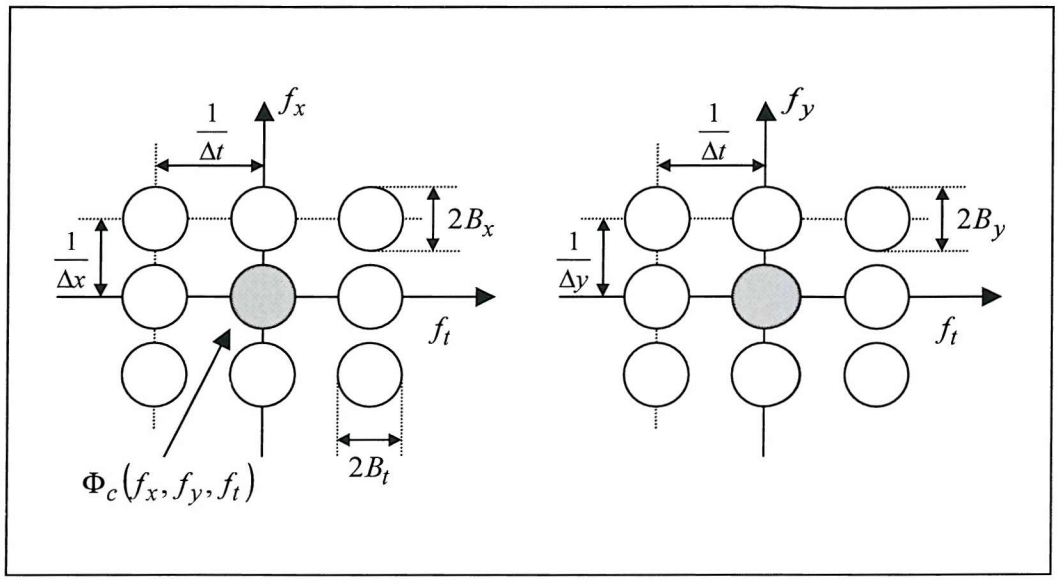

Figure 3. Three-dimensional spectrum of a video signal.

The uniqueness of the Fourier representation ensures that it is, in theory, possible to recover the original continuous signal, provided that the repetitions of $\Phi_c(f_x,f_y,f_t)$ of the continuous spectrum do not overlap, i.e. there is no aliasing in the process of sampling that leads to the discrete spectrum. This is expressed mathematically by Nyquist's theorem that states that perfect reconstruction is possible if the inverse of the sampling spacing in each direction is at least twice the bandwidth of the respective frequency. In other words, perfect reconstruction is possible if:

$$(\Delta x)^{-1}\geq 2B_x;\ (\Delta y)^{-1}\geq 2B_y;\ (\Delta t)^{-1}\geq 2B_t \tag{1.5}$$

In practice, the discrete signal is always an aliased version of the hypothetical, continuous one. This is because the physical device that samples the continuous signal, be it a photosensitive emulsion, a CRT camera or a CCD device, has less resolution (spatial and temporal) of the pattern of light that ultimately comprises the continuous signal. As a rather scholastic consideration, we can assume that the continuous image has a spatial resolution in the order of microns (10^{-6} metres), being this, the minimum size that visible light can discriminate without diffracting (about 10 times the visible light's wavelength range, $4\sim8\text{x}10^{-7}$ m). Such resolution is well beyond the capability of any image-capture device, including the human eye.

1.4.1. Spectrum of the interlaced video signal

The Fourier analysis can be equally applied to the interlaced video signal. However, this time the discrete video signal will have a displaced structure in the *y-t* plane. It is possible to show (Tekalp 1995) that the resulting spectrum will have a corresponding displaced structure, as shown in figure 4. The details of these results are not included here, but a discussion of Fourier spectra for arbitrary scanning formats can be found in appendix A.

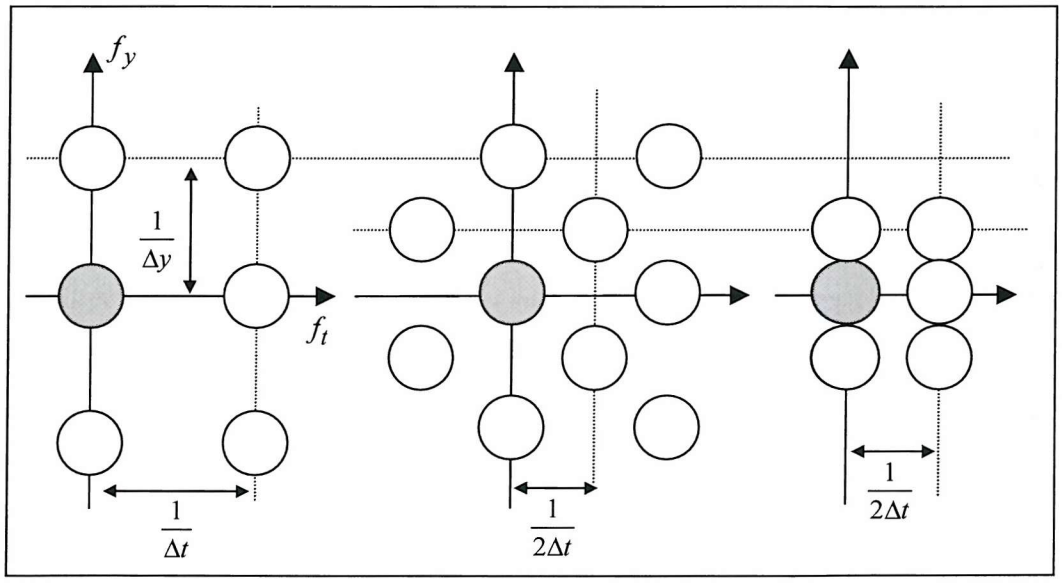

Figure 4. Progressive (left) and interlaced (centre) video spectra. The frame rate of the progressive signal is equal to the field rate of the interlaced signal. Right: spectrum of a single field sequence.

In figure 4 the field rate of the interlaced signal is equal to the frame rate of the progressive signal (see appendix A). This choice is made so that the progressive sequence (spectrum on the left side) is the signal we wish to obtain by de-interlacing the interlaced sequence (the spectrum on the right side).

From the figure, the non-linear nature of the de-interlacing problem can be seen. We can see how, in the interlaced format, the repetitions of the continuous spectrum are more closely packed than in the progressive format we wish to estimate. Therefore it is likely that, even if the target progressive spectrum satisfies (1.5), the interlaced spectrum might not. From the frequency domain it is easy to understand how the de-interlacing problem is strongly ill-conditioned.

In order to obtain the desired estimate from the interlaced field, the corresponding progressive frame should be band-limited, specifically its vertical and temporal single-side bandwidths should be limited respectively to $B_y \leq 1/2\Delta y$ and $B_t \leq 1/2\Delta t$. In simpler words, the progressive unknown signal should not have any vertical detail finer than two progressive, consecutive lines, and no significant time variation between two frames, that is to say that each odd field is substantially similar to its corresponding even field, and vice-versa. Under these conditions, perfect reconstruction is achievable by an ideal low-pass filter in the *y-t* plane. In other words, the de-interlacing problem is linear. Unfortunately, as we have discussed already, any reasonable video signal occupies most, if not all, of the available bandwidth, that is to say, it has detail on the finest scale available by the scanning format. Hence linear up-sampling will perform relatively poorly because of the constraints of Nyquist's theorem.

1.4.2. Effect of motion on the video spectrum

Another interesting aspect of frequency analysis is the effect of motion on the video spectrum. Suppose that there is an amount of vertical motion in the picture. For instance, there is an object that moves vertically with velocity v_y, in a otherwise static scene. Hence, the continuous video sequence with motion, $F_c^{(m)}(x,y,t)$, can be related to its static (without motion) version $F_c^{(s)}(x,y,t)$:

$$F_c^{(m)}(x,y,t) = F_c^{(s)}(x,y,t) * \delta(y - v_y t) \tag{1.6}$$

By performing a Fourier transform in the spatial variables, we obtain:

$$\tilde{\Phi}_c^{(m)}(f_x, f_y, t) = \tilde{\Phi}_c^{(s)}(f_x, f_y, t) \cdot e^{-i2\pi \cdot v_y \cdot f_y \cdot t} \tag{1.7}$$

where $\tilde{\Phi}_c^{(m)}$, $\tilde{\Phi}_c^{(s)}$ denote the partial Fourier transforms with respect to the spatial variables of the moving and static sequences respectively. By transforming both sides of (1.7) with respect to the temporal variable we finally obtain:

$$\Phi_c^{(m)}(f_x, f_y, f_t) = \Phi_c^{(s)}(f_x, f_y, f_t) \cdot \delta(f_t - 2\pi \cdot v_y \cdot f_y) \tag{1.8}$$

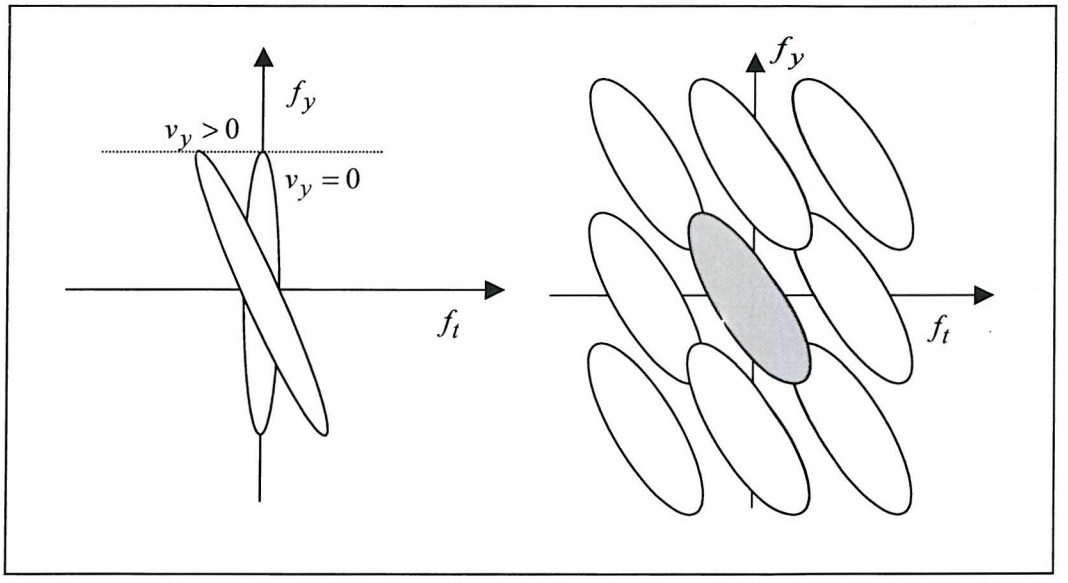

Figure 5. The effect of vertical motion on the continuous signal spectrum (left) and the discrete signal spectrum (right) The spectrum has been assumed elliptic in order to highlight the effect of motion.

Hence, the original static spectrum is sheered in the f_t direction by an amount that is linearly dependent on the value of f_y, as illustrated in figure 5. Note how motion can be another source of aliasing.

1.5. Linear interpolation and de-interlacing

The simplest way to realise an interpolation scheme is some form of linear combination of the sampling taps. However we have seen that in general the Nyquist's criterion is not satisfied by the video sequences both in spatial and temporal frequency domains. Linear interpolation does not address this problem and consequently linear filtering will only provide a simplified solution. Section 1.4 shows that in regions with little vertical detail (i.e. where the vertical alias is negligible), spatial linear filters can achieve a reasonably good interpolation. Conversely, for sequences with little inter-field motion, temporal filters can again obtain a satisfactory result. In which case linear interpolation can still play an important role in de-interlacing. Linear interpolation will be more completely discussed in the next chapter. In this section, we present two simple examples of the artefacts that can be generated by linear interpolation, whenever the bandwidth requirements for successful spatial/temporal linear interpolation are not satisfied. The demonstrations are performed separately in space and time.

As an illustration consider one of the simplest linear filter that can be employed, where the missing lines in the odd field are replaced by the average of the lines above and below the line being considered. A result using this scheme is shown in figure 6, where this simple algorithm has been applied to a 300x300-pixel test pattern. One can see that horizontal lines become blurred and oblique lines become stepped. The stepping of oblique lines is called *jagging*. The reduction of these jagging artefacts in spatial interpolation will be a main goal of this thesis. Note that an amount of jagging is already present in the original frame (left picture in figure 6). This is due to the low resolution chosen. One can see how the effect is much more pronounced in the de-interlaced frame (right picture in figure 6).

An alternative strategy is to use linear interpolation in the time domain, where the lines in the even field of a frame are replaced by the average of the corresponding lines of the previous and succeeding fields. Figure 7 illustrates some of the artefacts that can be introduced by such a routine. In this simulation the amount of movement has been purposely exaggerated, and it clearly exceeds the bandwidth limitations in

the temporal frequency domain. The result dramatically shows the form of artefact generated.

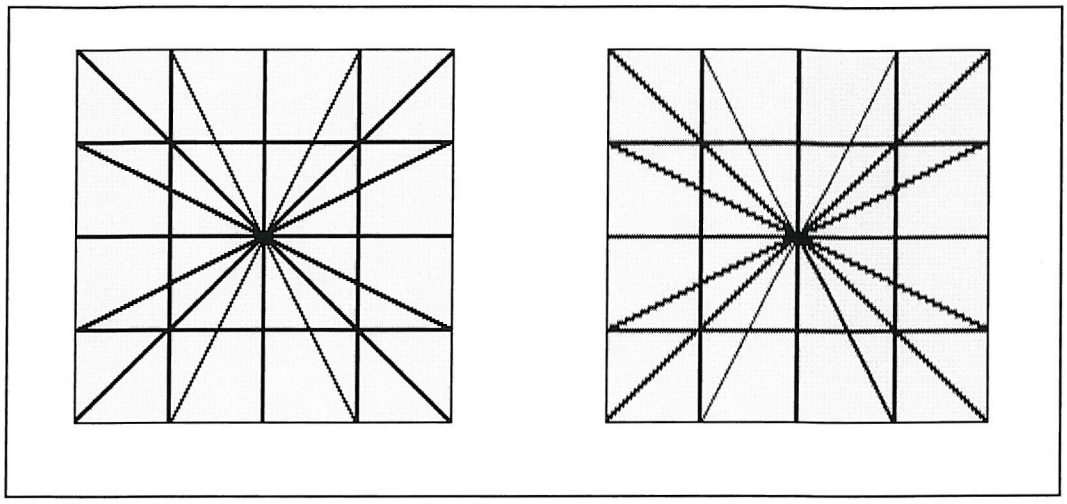

Figure 6. Spatial linear interpolation and "jagging" artefacts (right). 300x300 pixel test image. Note that the source frame (left) is already "jagged" on the oblique lines. This is due to the low resolution of the image. One can see how the jagging is much more evident in the de-interlaced frame (right) obtained by linearly up-sampling the even field of the source frame.

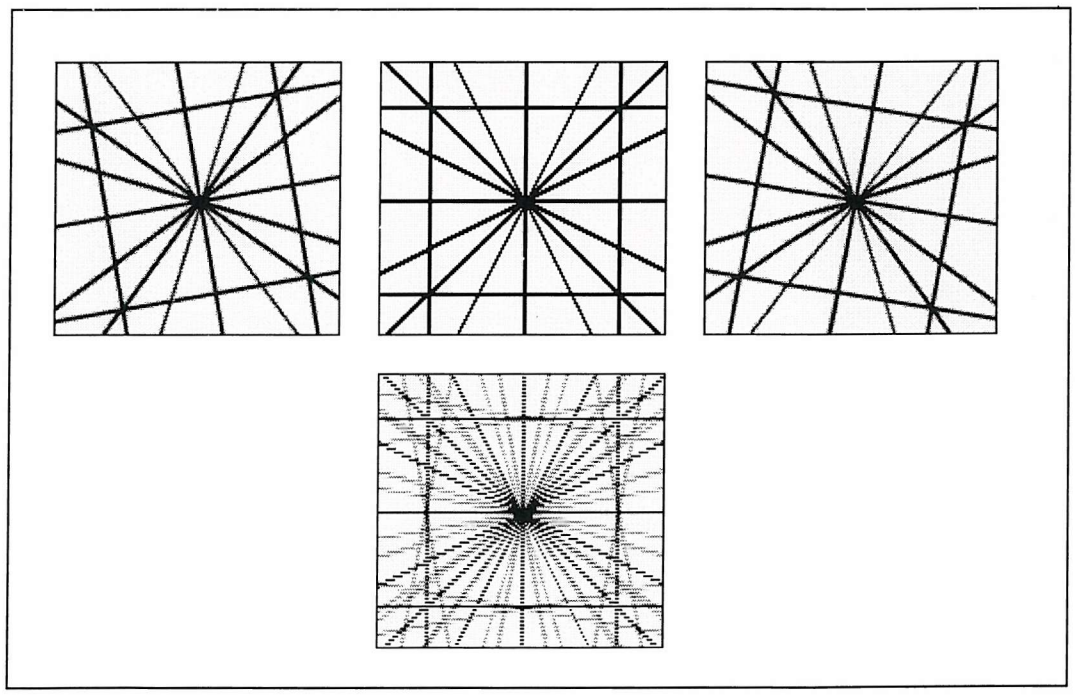

Figure 7. Non-motion compensated temporal linear interpolation, and motion artefacts (bottom). The even lines in the top-centre frame have been replaced by the average of the even lines in the previous frame (top-left) and the succeeding frame (top-right).

1.6. Non-linear interpolation and de-interlacing

An obvious step to overcome the limitations of linear interpolators is to use non-linear techniques. The basis of the interpolation is the same, since we use the information contained in a set of neighbouring pixels in the horizontal, vertical and possibly temporal directions. However, in the non-linear case the interpolation function is expressed as a *non linear combination* of these neighbouring pixel values (the sampling taps).

There are at least three arguments in favour of the non-linear approach. The first is based on simple considerations of the nature of the problem in the frequency domain. Any linear transformation of an arbitrary signal performs a linear weighting of the signal's spectrum. Some frequencies of the original spectrum are attenuated and some are amplified, with the phase usually experiencing some form of distortion, typically a phase shift in the transition bands of the spectrum. However, linear operations do not add new frequencies to the spectrum. For instance, a low-pass signal will remain low-pass, a band-pass signal will remain band-pass, and a high-pass signal will remain high-pass. Aliasing conversely is a form of distortion that changes the nature of the signal's spectrum, since it adds frequency bands that did not exist in the original signal, e.g. mapping high frequency content into the low-frequency band, and vice-versa. Many non-linear systems act in a similar way. A simple example is the multiplication of a signal by a cosine of frequency f_0 (modulation), which transforms low-pass signals into band-pass signals, and vice-versa. We assume that a proper non-linear system may take into account the non-linear distortion caused by aliasing and, once properly trained, could mitigate, if not completely reverse, it.

Another point in favour of the non-linear approach arises from geometrical interpretations. As we will see in the next chapter, interpolation can be described as a form of multi-dimensional mapping. Linear interpolators generate very simple mappings, insufficient to describe the complex input-output relationships occurring in the multi-dimensional input-output space. Non-linear techniques allow us to generate more complex and arbitrary shapes for this mapping.

The third reason in favour of the non-linear approach is of a statistical nature. The typical training procedure for a linear filter (learning-by-examples using a minimum mean squares error criteria, chapter 2) is such that the unavoidable error between the estimated output and the test output will be statistically orthogonal to the input. This kind of estimate can be considered satisfactory when the distribution in the input-output space has Gaussian statistics. This is a requirement that is not satisfied by natural images (Petrou, Bosdogianni 1999). Non-linear techniques are able to deal with more complex statistical relations between the error and the input by taking account of the higher-order moments of the input distribution (Collis, 1997).

In this thesis, we will study two non-linear techniques: the Volterra series and the Radial Basis Function Networks. Volterra series is a mathematical tool that has been intensively studied to model non-linear systems (Schetzen, 1989). We can describe Volterra series as a polynomial (Taylor) series with finite memory. The application of Volterra series to de-interlacing has been studied by Collis et al. (1997), with interesting results in terms of the output quality. Being polynomial, Volterra series generate mappings far more complex and flexible than the linear interpolator does. Furthermore, it has been shown that a Volterra series, trained using a learning-by-examples procedure, is able to uncorrelate the error not only with the input, as in the linear case, but with higher-order combinations of the input elements. In fact, the linear filter is a special case of Volterra series, specifically a series of first order. These properties come from the strong bond Volterra series have with Higher Order Spectra theory (Schetzen, 1989), as we will see in chapter 2.

Radial Basis Function Networks (RBFN) (Powell, 1987) is another technique that has been applied to a variety of non-linear problems. RBFN generate a greater range of input-output mappings than Volterra series. Furthermore, RBFN have a strong theoretical bond with *statistical density estimation*, notably with Gaussian Mixture models (see chapter 3). The study of RBFN applied to de-interlacing will constitute the primary subject of this work, and chapter 3 will be dedicated solely to the description of RBFN in a more general context.

Linear filters, Volterra series and RBFN are specific examples of the more general structure of *single-layer feed-forward networks*. These structures are part of an even

larger family of networks, the multi-layer feed-forward networks. Compared to the latter, single-layer networks have distinct advantages (and disadvantages) that will be discussed in chapter 2. A schematic of a general single-layer feed-forward network is shown in figure 8.

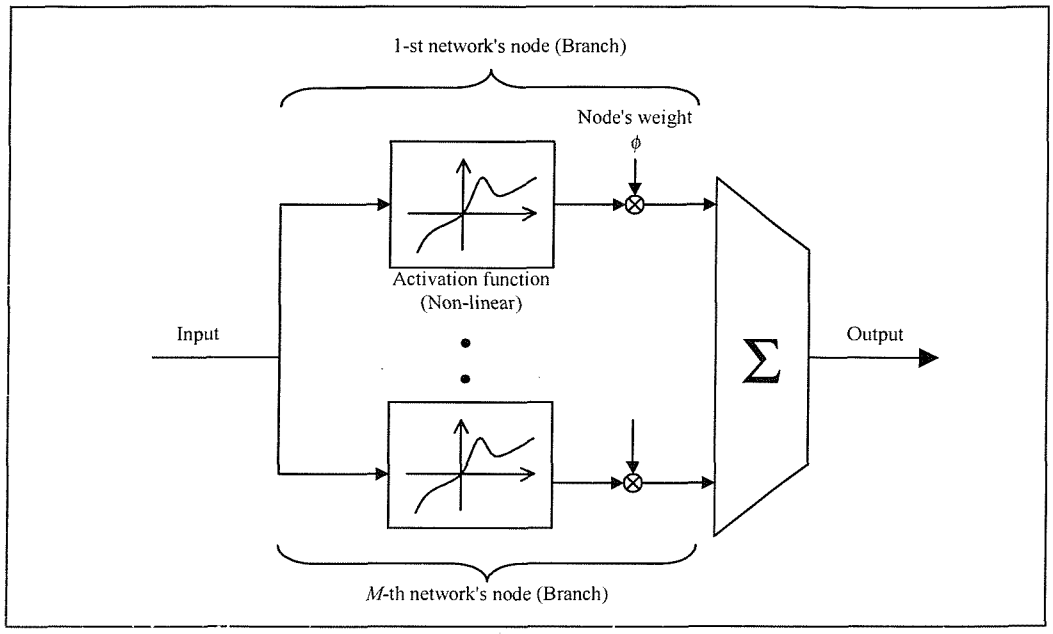

Figure 8. Schematic diagram of a single-layer network.

The main disadvantage of the non-linear approach is the computational effort required, that often makes non-linear techniques impractical. Consider a 25 frames/second, 625 lines/frame video sequence, and a horizontal resolution equal to the vertical resolution, so that there are 625 columns per frame. One then has 25 x 625 x 625/2=4.5 x 10^6 missing pixels per second to estimate. Hence the computational lag must be limited to the order of tenths of microseconds. Such time scales require the filter to be implemented in hardware. Non-linear filters, with their relatively large computational burden, may lead to impractical expensive hardware. Appendix B is focused on the study of computational requirements of the two non-linear techniques studied in this thesis.

1.7. Conclusions

In this chapter the de-interlacing problem has been briefly introduced, together with the fundamental engineering problems related to it. De-interlacing is a form of interpolation and as such, the possibility of recovering the missing information is limited by the Nyquist's theorem. The performance of linear spatial-temporal filters is conditioned by the frequency content of the image. Typical video signals occupy the available bandwidth well beyond the Nyquist's limit and the up-sampling problem is ill-posed in those regions of the video sequence where rapid variations in the vertical direction and in time occur. This results in a series of characteristic aliasing artefacts. An example of spatial aliasing is the "jagging" artefact, where the alternation of blurred lines (the interpolated lines) and sharp lines (the original lines) create a stepping effect on oblique, high contrast edges. Nevertheless, linear de-interlacing often provides satisfactory results and in fact it is a common choice for consumer electronics.

In order to increase the performance of linear de-interlacing, non-linear interpolation may be attempted. Non-linear systems are capable of yielding a much richer functional relationship between the sampled input and the target output. In particular, three aspects of the non-linear approach are interesting in terms of de-interlacing: the ability of generating new frequency content beyond the Nyquist's limit, the increased arbitrariness of the multi-dimensional mapping, the superior statistical modelling of the input-output pattern.

Possible choices for the non-linear architecture are the Volterra series and the Radial Basis Function Networks. These two techniques belong to the larger family of single-layer feed-forward networks, that are commonly used in signal processing and that show interesting properties in terms of training and performance, as it will be shown in the next chapter. A point to remember is that the increased performance given by non-linear techniques comes with an increase in the computational complexity. Therefore one should be careful to trade-of accuracy with computational convenience.

2. SINGLE-LAYER NETWORKS AND INTERPOLATION

2.1. Introduction

In the previous chapter, we have introduced the motivation that lays behind the quest for de-interlacing systems. De-interlacing can be described as an interpolation task, being a form of up-sampling. In order to estimate each pixel in the missing lines that transform a field into a frame, we try to realise a suitable function of a set of neighbouring pixels in the corresponding known field (intra-frame de-interlacing) and in the previous and successive known fields (inter-frame de-interlacing). Specifically we try to realise a mapping $\mathbf{x} \rightarrow y(\mathbf{x})$ from a multidimensional input space of vectors $\mathbf{x} = [x_1 \ldots x_D]^T$ (the vector of samples, where D is clearly the number of taps in the sampling lattice) to a one-dimensional output space y. For colour pictures, the output is three-dimensional, representing the three colour components. However, in the general case the de-interlacing problem for colour pictures can be separated into three one-dimensional output mappings. Thus, the main task of de-interlacing is to find an appropriate function $y = f(\mathbf{x})$ that realises a satisfactory estimate of the missing lines via such a mapping. We can call this function the *model* of our de-interlacing system.

Unfortunately linear models for de-interlacing perform poorly, since the nature of the de-interlacing problem is non-linear. The task of finding an appropriate non-linear model is rather more difficult than the task of finding a linear one, for reasons that will be discussed in more detail later. Linear models benefit from a broad collection of analytical tools that help to identify appropriate solutions in a computationally efficient manner. Conversely, non-linear techniques often do not benefit from an exact solution, and usually are computationally demanding. However, the point to remember here is that the task of finding a model for a de-interlacing system, linear or non-linear, is the task of finding a multidimensional mapping.

In this chapter, we will now discuss the properties of a particular family of models that can realise such mappings, namely the family of *single-layer networks*. Single-layer networks can implement both linear and non-linear mappings. The principal task

of this thesis is to produce a successful non-linear de-interlacing system using a particular single-layer network called the *Radial Basis Function Network* (RBFN).

Prior to the description of the basic theory of single-layer networks, we introduce the concepts of *parametric models*, *learning by examples* and *cost functions* that form the basis of the aforementioned theory.

2.2. Parametric models

A model is defined as parametric whenever its input-output function $y(\mathbf{x}) = f(\mathbf{x})$ contains a number of adjustable parameters that change the mapping $\mathbf{x} \rightarrow y(\mathbf{x})$. This can be expressed as $y(\mathbf{x}) = f(\mathbf{x};\Phi)$, where $\Phi = [\phi_1 \ldots \phi_M]^T$ is a vector formed by the M parameters of the model (the reason why the first element of Φ is represented with subscript 1 will be explained in the following sections, when the *bias term* ϕ_0 will be introduced). The determination of the best set of parameters, i.e. a set of parameters that achieves a determinate goal, is known as *parameter optimisation.* In many cases, such a problem is non-linear. Non-linear behaviour may arise from the non-linear dependence of $f(\mathbf{x};\Phi)$ on Φ, or from the non-linear nature of the particular goal we have to achieve. The latter is usually expressed in terms of a *cost function.* By cost function we mean a function $C(\Phi)$ whose values are high when the model performs poorly, and is low when the system performs well. Note that we have expressed the cost function as dependent on a set of variables, still unspecified. However, in a parametric model $C(\Phi)$ is necessarily a function of the model's parameters Φ. Thus, the main task of determining an appropriate set Φ is that of finding an optimal set of parameters that minimises the cost function.

In determining the appropriate model to describe a system, a possible approach is to derive an analytical expression based on a prior knowledge or assumptions on the system itself. This is, for instance, the case in many mechanical or electronic systems, where we model the system in terms of masses, forces etc, or conversely resistors, capacitors, inductors and other physical components of the system under investigation. This particular model is often referred as *transparent box* since the

parameters and the functional structure of the model give an interpretation in terms of the physical system.

If the information necessary to create a transparent box model is not available, an alternative is to describe a very general parametric model with enough degrees of freedom to generate a satisfactory input-output mapping, without a specific relation with the physical system (*black box*).

2.3. Learning-by-examples training

Regardless of the choice of the model, we have to face the task of identifying the suitable values for the parameter set $\mathbf{\Phi}$. In many cases, the only information available is that a given sequence of L inputs $\mathbf{X}=[\mathbf{x}_0 \dots \mathbf{x}_{L-1}]^T$ generates a corresponding sequence of *targets* $\mathbf{T}=[t_0 \dots t_{L-1}]^T$. The most reasonable way to estimate the system is then to use a model that is able to generate an output sequence $\mathbf{Y}(\mathbf{X})=[y_0 \dots y_{L-1}]^T=[y(\mathbf{x}_0)\dots y(\mathbf{x}_{L-1})]^T$ that matches (in a sense that will be clarified later) the sequence $\mathbf{T}$, once the input sequence has been fed into its input. This procedure is often referred as *learning-by-examples*. We assume that the observed input-target sequence is representative of the problem by somehow representing all the possible input sequences. The input-target sequence $\{\mathbf{X},\mathbf{T}\}=\{\mathbf{x}_n,t_n\}_{n=0\dots L-1}$ is called the *training set*.

When the model is parametric, the training set must be used somehow to determine the model's parameters. In other words, scope of the training procedure is to adjust the model's parameters so that the model fits the training set in terms of the cost function $C(\mathbf{\Phi})$. In other words, the task of the training procedure is to find the set $\mathbf{\Phi}^*$ of parameters such that:

$$\mathbf{\Phi}^* = arg\left\{ \min_{\mathbf{\Phi}} \left[C(\mathbf{\Phi} \,|\, \mathbf{T},\mathbf{X}) \right] \right\} \qquad (2.1)$$

Figure 1 shows a schematic diagram of the learning-by-examples training procedure. The choice of the parametric model determines *the residual cost* $C(\Phi^*)$, that is usually not null, and thus the choice of the model is a key point of our analysis.

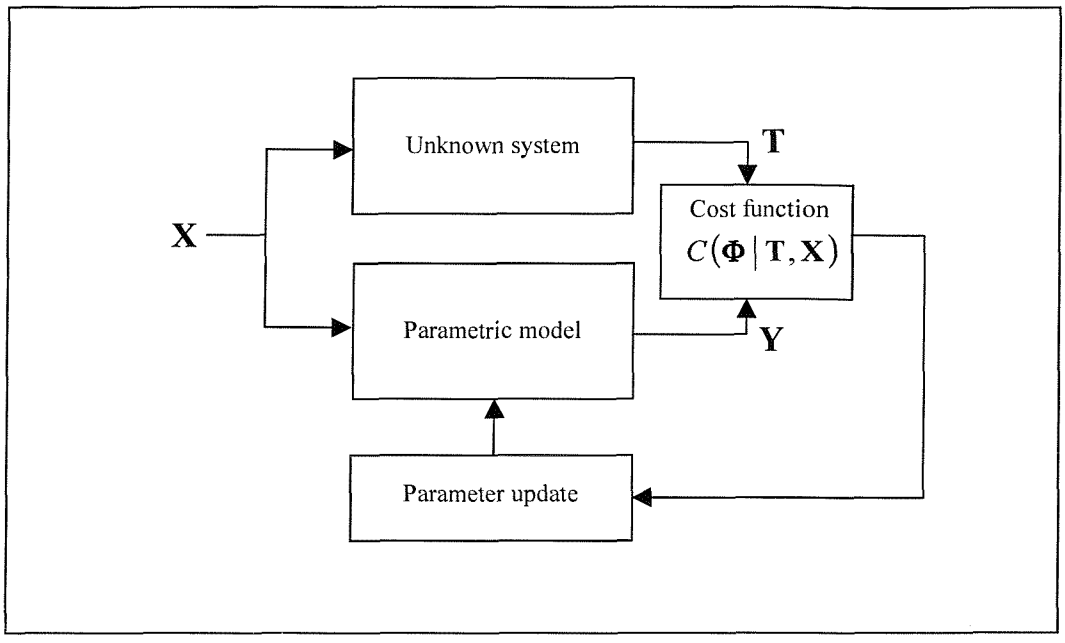

Figure 1. Learning-by-examples and cost function

2.3.1. Size of the model and generalisation

As has already been discussed, the choice of an appropriate model structure is crucial to produce a proper representation of the unknown system. In the case of a parametric model trained using a learning-by-examples procedure, this choice can basically be divided in two sub-problems: choice of the number of parameters in the model, and choice of the mathematical relations that link these parameters to the model output. In this section we will focus on the first of these two problems.

In learning-by-examples training, there is a trivial way to generate a parametric model that exactly matches the training set: a look-up table (LUT). In simple words, a LUT is a memory bank that associates, to any of its memory locations, a corresponding value. In a LUT each memory location's address represents an input to the function, and the corresponding value in the table represents the output. With such a model, the training problem is trivial because we can store the full training set in the LUT, where the sequence $\mathbf{T}$ can be viewed as the parameter of the model, and is exactly matched

for any input belonging to the training set. This approach leads to impractical methods for modelling complex systems for two reasons (Girosi, Poggio 1989).

Firstly, as the number L of training patterns increases, the size of the LUT becomes unrealistic. This is often referred as the *curse of dimensionality* (Bellman 1961). Secondly, and more importantly, there is a subtle difference between exact interpolation and learning. Both cases can be solved using a learning-by-examples strategy; however, in the former case we want to achieve the best match possible between the model's output $\mathbf{Y}$ and the system's output $\mathbf{T}$. Conversely in learning we want our model to describe the data generator that lies behind the training set (Bishop, 1995), rather than the particular, incomplete realisation of such generator given by the training set. Hence, exact interpolation is generally not desirable, since the model must be able to correctly estimate input-target patterns that are not included in the training set.

Excessive accuracy in interpolation is often referred as over-fitting. A model that fits the training set with excessive accuracy will generally fail to produce accurate estimates of patterns not included in the training set. This unwelcomed behaviour is amplified when the training set is affected by noise that generates incorrect patterns. An over-fitted model will interpolate these noisy patterns and will produce a poor performance.

In order to reduce over-fitting in parametric models, an accurate choice of the size (number of parameters) of the model is of paramount importance. The problem of selecting an appropriate size for a parametric model is a classic example of *Occam's razor* (Domingos 1999). The principle is that we prefer simple models to more complex models, and this preference must be traded off against the accuracy of the model in fitting the training set. The ability of a model to predict unseen patterns is called *generalisation*. Chapter 7 of this thesis will consider this problem in detail.

2.4. Cost functions

To operate a learning-by-examples strategy one must provide a suitable cost function. This function must produce a small outcome when the model's output matches the target sequence and a large outcome when it does not.

This section mainly discusses a particular kind of cost function, the *Mean Squared Error* (MSE). This cost function presents some useful advantages, noticeably the fact that it leads to simple training algorithms, and has a strong statistical background. Some alternatives to this cost function will be presented.

From now on, the model will be considered parametric. Hence the cost function will depend on the model's parameters. The goal of the training algorithm is to produce a mapping with the minimum MSE (MMSE).

2.4.1. Mean Squared Error

Among different cost functions used in applications, MSE is by far one of the most popular. Given the parametric model $y(\mathbf{x}) = f(\mathbf{x};\mathbf{\Phi})$, the MSE for the training set $\{\mathbf{X},\mathbf{T}\}$ is given by:

$$C(\mathbf{\Phi}) = \frac{1}{L}\sum_{n=0}^{L-1} e_n^2 = \frac{1}{L}\sum_{n=0}^{L-1}[y(\mathbf{x}_n;\mathbf{\Phi}) - t_n]^2 \tag{2.2}$$

The term $1/L$ is introduced in order to make meaningful comparisons between training set of different size L. For an infinite training set ($L \to \infty$), equation (2.2) can be written as (Bishop, 1995):

$$\begin{aligned}\lim_{L\to\infty} C(\mathbf{\Phi}) &= \lim_{L\to\infty} \frac{1}{L}\sum_{n=0}^{L-1}[y(\mathbf{x}_n;\mathbf{\Phi}) - t_n]^2 = \\ &= \int [y(\mathbf{x};\mathbf{\Phi}) - \langle t \,|\mathbf{x}\rangle]^2 \, p(\mathbf{x})d\mathbf{x} + \int \left[\left\langle t^2 \,\middle|\mathbf{x}\right\rangle - \langle t \,|\mathbf{x}\rangle^2\right] p(\mathbf{x})d\mathbf{x}\end{aligned} \tag{2.3}$$

Where $\langle t|\mathbf{x}\rangle$ is the conditional average of t over the set $\mathbf{x}$, and $p(\mathbf{x})$ is the probability density function of the input. The second term in the right side of (2.3) does not depend on the parameters and can be neglected. The first term vanishes if

$$y(\mathbf{x};\Phi) = \langle t|\,\mathbf{x}\rangle \tag{2.4}$$

Hence the MMSE solution is the value Φ^* that satisfies (2.4). Equation (2.4) is a key result as it describes the optimal network mapping in terms of a conditional average of the data, i.e. as the *regression* of target data t conditioned on $\mathbf{x}$. This result is illustrated graphically in figure 2 for a single dimensional case, and assuming $p(t|\mathbf{x})$ Gaussian for simplicity.

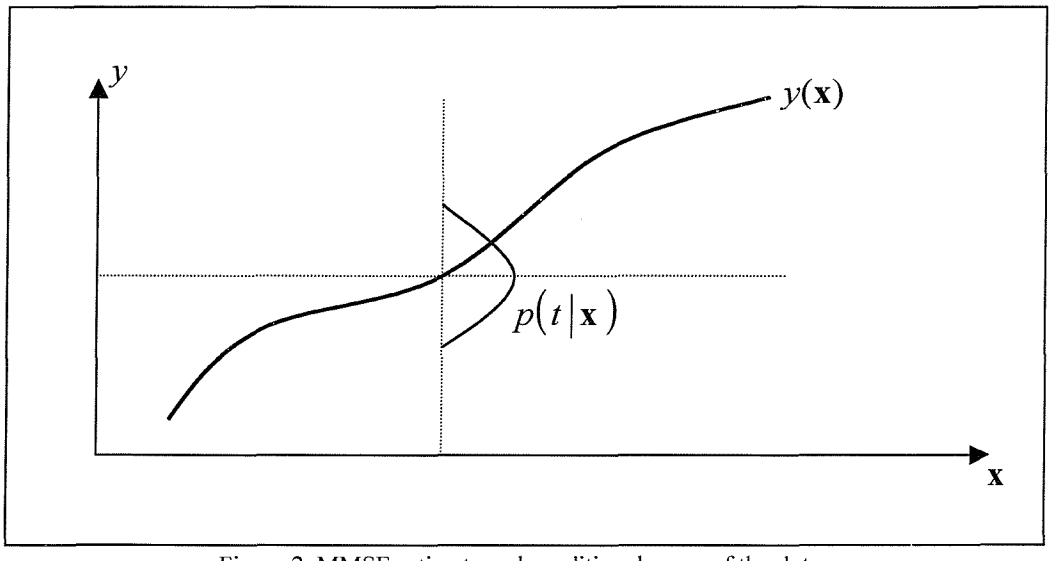

Figure 2. MMSE estimate and conditional mean of the data.

The residual error $\varepsilon = e(\Phi^*)$ is the second term at the right hand side of (2.3) evaluated for $\Phi=\Phi^*$, and can be written as the average of a variance function $s^2(\mathbf{x})$:

$$s^2(\mathbf{x}) = E\left[\sigma^2(\mathbf{x})\right] = \int \sigma^2(\mathbf{x})\, p(\mathbf{x})\, d\mathbf{x} \tag{2.5}$$

Where $\sigma^2(\mathbf{x}) = \langle t^2\,|\mathbf{x}\rangle - \langle t\,|\mathbf{x}\rangle^2$ is the output variance of t conditioned on the input $\mathbf{x}$. Equations (2.4) and (2.5) are the quantities that we can determine by

minimising the MSE. Hence, the MMSE mapping cannot discriminate two training sets with the same conditional mean (2.4) and conditional average variance (2.5).

Note that the input density $p(\mathbf{x})$ weights both integrands on the right hand side of (2.3). From the first term, we infer that the network function incurs little cost to departures from the conditional average in regions where the density is small.

The key consequence is that we must consider the *representativeness* of the input set. If we desire a small error in a subset $\mathbf{X}_1$ of the input sequence, the subset density $p(\mathbf{x} \,|\mathbf{x} \in \mathbf{X}_1)$ must be reasonably high. The training must be reasonably representative of the general, real behaviour of the input. However, we may want an accurate model of the system for input-output patterns that rarely occur in real situations and/or in the training set, but nevertheless are critical in terms of performance if not described accurately. Hence, we must compensate the density function of the training set in order to make these patterns sufficiently represented.

Different attempts have been made to compensate for poor performance over low-density subsets, and more generally to account in the cost function features that are insufficiently represented by the density function. An example, applied in image processing, can be found in Tompa *et al.*(2000), where density function is compensated according to some measure of perceptual relevance, i.e. some measure of the observer's perception. De Stefano *et al.* (2000) analyse a Frequency Weighted MSE in order to emphasise the performance in the bands where the human visual system is more sensitive.

2.4.2. Linear MMSE training

Learning-by-examples MMSE training benefits of a simple solution $\mathbf{\Phi}^*$ if the model input-output mapping is linear, i.e. given by a weighted sum of the input vector's elements:

$$y = \sum_{j=1}^{D} \phi_j x_j + \phi_0 = \mathbf{\Phi}^T \mathbf{x} + \phi_0 \qquad (2.6)$$

Such a model is usually known as *Wiener filter* (Oppenheim, Schafer 1974), or as moving-average (MA) as the output is the result of a weighted average of the input samples. The elements of the parameter vector are denoted as *weights*. Note that we have introduced a constant parameter ϕ_0, the bias term, briefly introduced in section 2.2.

Substituting (2.6) in (2.2) and neglecting the scaling factor L we obtain

$$C(\hat{\mathbf{\Phi}}) = \sum_{n=0}^{L-1} t_n^2 + \hat{\mathbf{\Phi}}^T \left(\sum_{n=0}^{L-1} \hat{\mathbf{x}}_n \hat{\mathbf{x}}_n^T \right) \hat{\mathbf{\Phi}} - 2\hat{\mathbf{\Phi}}^T \sum_{n=0}^{L-1} t_n \cdot \hat{\mathbf{x}}_n^T \tag{2.7}$$

where the bias term has been appended to the parameter (weight) vector and consequently a unitary factor has been appended to the input vector:

$$\hat{\mathbf{\Phi}}^T = \begin{bmatrix} \phi_0 & \mathbf{\Phi}^T \end{bmatrix} \qquad \hat{\mathbf{x}}^T = \begin{bmatrix} 1 & \mathbf{x}^T \end{bmatrix} \tag{2.8}$$

Equation (2.7) can be expressed in a more compact form as:

$$C(\hat{\mathbf{\Phi}}) = \mathbf{T}^T \mathbf{T} + \hat{\mathbf{\Phi}}^T \mathbf{R}\, \hat{\mathbf{\Phi}} - 2\mathbf{P} \tag{2.9}$$

where $\mathbf{R}$ is the correlation matrix of the input, and $\mathbf{P}$ is the input-output cross-correlation vector:

$$\mathbf{R} = \sum_{n=0}^{L-1} \hat{\mathbf{x}}_n \cdot \hat{\mathbf{x}}_n^T = \mathbf{X}^T \mathbf{X} \qquad \mathbf{P} = \sum_{n=0}^{L-1} t_n \cdot \hat{\mathbf{x}}_n^T = \mathbf{X}^T \mathbf{T} \tag{2.10}$$

Equation (2.9) is quadratic in $\hat{\mathbf{\Phi}}$, and its solution can be easily found solving

$$\frac{\partial C}{\partial \hat{\mathbf{\Phi}}} = 0 \;\Rightarrow\; \mathbf{R} \cdot \hat{\mathbf{\Phi}} - \mathbf{P} = 0 \tag{2.11}$$

Assuming $\mathbf{R}$ is non-singular the solution can be written as:

$$\hat{\mathbf{\Phi}}^* = \mathbf{R}^{-1} \cdot \mathbf{P} \tag{2.12}$$

It is equally easy to demonstrate that:

$$\frac{\partial C}{\partial \phi_0} = 0 \Rightarrow \phi_0 = \frac{1}{L}\sum_{n=0}^{L-1} t_n - \frac{1}{L}\sum_{n=0}^{L-1} \mathbf{\Phi}^T \cdot \mathbf{x}_n \tag{2.13}$$

Therefore the bias accounts for the difference between the average value of the weighted sum of the input elements and the mean value of the target t. For simplicity, we will commonly refer to $\mathbf{\Phi}^*$ (rather than $\hat{\mathbf{\Phi}}^*$) implicitly assuming the presence of the bias term.

Problems arise in solving (2.11) when $\mathbf{R}$ is singular, or the calculation of $\mathbf{R}^{-1}$ *ill-conditioned* (i.e. the eigenvalues of $\mathbf{R}$ span a large dynamic range). Techniques like *singular value decomposition* (SVD) and *regularisation* (Golub 1970) can be used to find a suitable solution in these cases.

One property of the linear MMSE estimate that can be easily inferred from above is the *orthogonality principle*, that states that the residual MMSE error $e^* = e(\mathbf{\Phi}^*)$ is statistically orthogonal to the input $\hat{\mathbf{x}}$. It is easy to show that:

$$E\left[e^* \mathbf{x}^T\right] = E\left\{\left[t - (\mathbf{\Phi}^*)^T \mathbf{x}\right]\mathbf{x}^T\right\} = 0 \tag{2.14}$$

Equation (2.14) is a statement of orthogonality for the optimal solution. The one-dimensional space spanned by the error is orthogonal to the multi-dimensional space spanned by the input.

2.4.3. MMSE and Maximum Likelihood estimate

Assuming that the target sequence is the sum of a deterministic sequence t_n^* and a zero mean Gaussian stochastic process ε_n,

$$t_n = t_n^* + \varepsilon_n \tag{2.15}$$

it is possible to demonstrate (Van Trees 1971) that the MMSE solution $\mathbf{\Phi}^*$ is also the *Maximum-Likelihood* (ML) solution, maximising the maximum-likelihood function:

$$L(\mathbf{\Phi}) = p(\mathbf{T} \mid \mathbf{X}, \mathbf{\Phi}) \tag{2.16}$$

Equation (2.16) corresponds to the intuitive idea of choosing the parameter $\mathbf{\Phi}$ which is most likely to give rise to the observed target, given the observed input. A more formal discussion of the origins of the ML procedure is given by Akaike (1973).

2.4.4. Minkowski-*R* error

As an example of an alternative cost function, we now discuss the *Minkowski-R error*:

$$C(\mathbf{\Phi}) = \frac{1}{L} \sum_{n=0}^{L-1} | y(\mathbf{x}_n ; \mathbf{\Phi}) - t_n |^R, \quad R > 0 \tag{2.17}$$

The Minkowski-*R* error function gives the ML estimate for a more general distribution of ε_n compared to (2.15), given by (Bishop, 1995):

$$p(\varepsilon_n) = \frac{R \beta^{1/R}}{2\Gamma(1/R)} exp\left(-\beta |\varepsilon_n|^R \right) \tag{2.18}$$

Where Γ is the *Gamma function*

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} \, dt \tag{2.19}$$

Note that (2.17) reduces to the MSE function and (2.18) to the Gaussian distribution if $R=2$. If $R=1$, the distribution (2.16) becomes a *Laplacian* distribution. In this case the ML estimate is less sensitive to *outliers*, i.e. isolated small groups of inputs that sensibly depart from the rest of the input set and could dominate the MMSE solution by virtue of their large value. On the other hand, more accuracy on the outliers could be achieved if $R>2$. The obvious drawback of the Minkowski-R error compared to the MMSE estimate is that it does not benefit of a simple solution like (2.12).

This section highlights three key requirements for successful learning-by-examples training. Firstly, the training set must be sufficiently representative of the unknown system's behaviour. This usually leads to a lower bound in the size of the training set. Secondly, the model function $f(\mathbf{x};\mathbf{\Phi})$ must be sufficiently general to describe the complexity of the system. Usually this determines a lower bound for the complexity of the network. The third assumption is that we are able to achieve a reasonable minimisation of the cost function. In other words, the cost function $C(\mathbf{\Phi})$ must be simple enough to allow the global minimum to be found.

2.5. Single-layer networks

We now analyse a parametric model, called *single-layer networks*, which forms the basis for the techniques we will use in this work. In this section we introduce some preliminary concepts, which are generally shared by all the particular realisations of this architecture. The general mathematical formulation of a single layer network with M *nodes*, or branches, is the expression:

$$y(\mathbf{x}_n) = \phi_0 + \sum_{j=1}^{M} \phi_j \cdot h_j(\mathbf{x}_n) \qquad (2.20)$$

The element $\phi_j \cdot h_j(\mathbf{x}_n)$ is called the j-th node of the network, where $h_j(\mathbf{x}_n)$ is a generic, possibly non-linear, *node activation function* $R^D \rightarrow R$, and ϕ_j is called the weight of the node j.

The model expressed by (2.20) is an evident generalisation of the linear model described in (2.6). It is therefore straightforward to compute the MMSE estimate of the parameters

$$\mathbf{\Phi}^* = \left[\left(\mathbf{H}^T \cdot \mathbf{H} \right)^{-1} \cdot \mathbf{H}^T \right] \cdot \mathbf{Y} = \mathbf{R}^{-1} \cdot \mathbf{P} \qquad (2.21)$$

Where $\mathbf{H} = [\mathbf{h}_0 \ldots \mathbf{h}_{L-1}]^T$, $\mathbf{h}_n = [1 \; h_1(\mathbf{x}_n) \; \ldots \; h_M(\mathbf{x}_n)]^T$ and the matrices $\mathbf{R}$ and $\mathbf{P}$ are analogous to those in (2.12), with $\mathbf{H}$ taking the role of $\mathbf{X}$. The matrix $\mathbf{H}^+ = \left(\mathbf{H}^T \cdot \mathbf{H} \right)^{-1} \cdot \mathbf{H}^T$ is known as the Moore-Penrose *pseudo-inverse* of $\mathbf{H}$ (Penrose, 1955), and provides a way to solve a linear system when the number of unknowns is different from the number of equations. If $M>L$ (under-determined system, infinite number of solutions), (2.21) provides the solution $\mathbf{\Phi}^*$ with the smallest norm. If $M<L$ (over-determined system), (2.21) provides the MMSE solution.

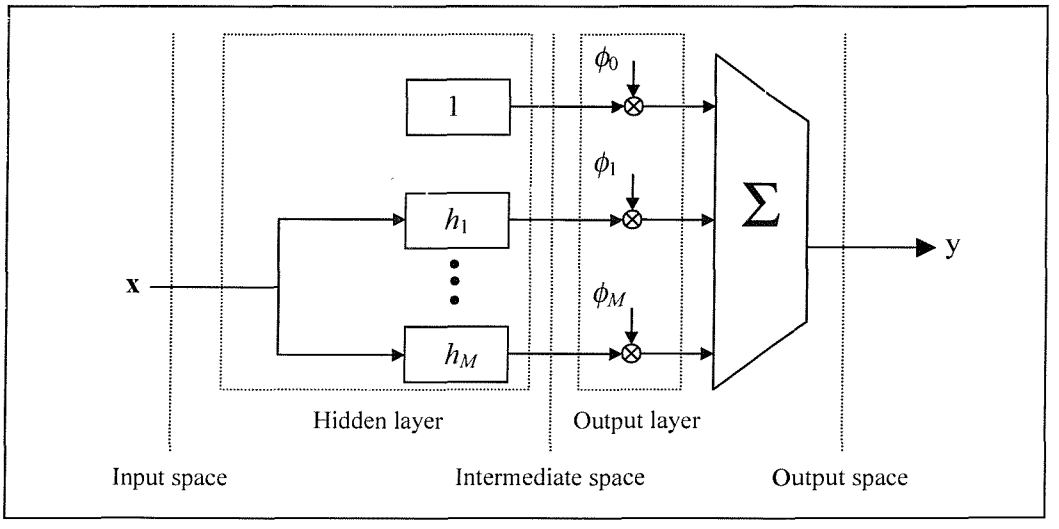

Figure 3. Two-layer formalism of the single-layer network.

Single-layer networks belong to the larger family of *feed-forward networks*, characterised by having no feed-back loops or lateral node connections. The single-layer structure is depicted in figure 3.

One may note that in the figure we use a two-layer formalism. In literature the linear weights are commonly considered part of the node, rather than being part of a separate layer in series with the non-linear function, as depicted in figure 3. One advantage of the single-layer formalism is to exploit the similarity between single layer networks and *Gaussian mixtures*, described in chapter 3. In this model, the weight j is considered as the prior probability of a class j having generated the input $\mathbf{x}$, while $h_j(\mathbf{x})$ is the density of $\mathbf{x}$, supposed to be generated by that class. Hence a weight and its corresponding activation function are closely related and considered as part of the same node.

Nevertheless, the two-layer formalism gives us a more precise description of the mapping obtained. It is clear that (2.20) is the combination of a mapping $\mathbf{x} \in R^D \rightarrow \mathbf{h} \in R^M$, typically non-linear, and a linear mapping $\mathbf{h} \in R^M \rightarrow y \in R$. It is possible to identify an *intermediate space* $\mathbf{H} \subset R^M$.

Therefore the linear weights have the status of nodes of the $R^M \rightarrow R$ linear mapping. Consequently, the non-linear layer should generate a $R^D \rightarrow R^M$ mapping that makes the $R^M \rightarrow R$ mapping as "linearly solvable" as possible.

2.5.1. Non-linear parameters

The network described by (2.20) is usually referred as a *linear-in-the-parameters* network (Duda, Hart 1973). The extension of the model to include non-linear parameters (i.e. parameters in the non-linear activation functions) is straightforward. In this way we add more degrees of freedom that can be used to find a suitable solution to our problem and achieve a more satisfactory residual error in equation (2.6).

Unfortunately, the cost we pay is that the MMSE solution is non-linear. Therefore some form of *non-linear optimisation* is generally required. This leads to the problem of finding the global minimum of a cost function that potentially contains large number of local minima.

2.6. The Volterra series

The linear interpolator described by (2.6) is the simplest example of single-layer network, where the intermediate space is equal to the input, $\mathbf{H}(\mathbf{X}) = \mathbf{X}$. Equation (2.6) is the discrete version of the continuous linear filter expressed by the convolution:

$$y(t) = \int_{-\infty}^{t} v(\tau) \cdot x(t-\tau) d\tau \tag{2.22}$$

where $v(\tau)$ is the *impulse response* of the linear filter (assumed here to be causal). Volterra (1959) was the first to study a powerful, non-linear extension of the convolution integral in (2.22) by introducing a series of high-order, multi-linear operators defined by multiple convolutions

$$\begin{aligned} y(t) = & \int_{-\infty}^{t} v^{(1)}(\tau) \cdot x(t-\tau) d\tau + \int_{-\infty}^{t} \int_{-\infty}^{t} v^{(2)}(\tau_1, \tau_2) \cdot x(t-\tau_1) \cdot x(t-\tau_2) d\tau_1 d\tau_2 + \\ & \cdots + \int_{-\infty}^{t} \int_{-\infty}^{t} \cdots \int_{-\infty}^{t} v^{(p)}(\tau_1, \ldots, \tau_p) \cdot x(t-\tau_1) \cdot \cdots \cdot x(t-\tau_p) d\tau_1 \cdots d\tau_p + \cdots \end{aligned} \tag{2.23}$$

The element $v^{(p)}(\tau_1, \ldots, \tau_p)$ is called the *Volterra kernel* of order p of the functional series (2.23). Clearly the impulse response $v(\tau)$ in (2.22) is the Volterra kernel of first order. The discrete-time, *finite-memory* version of (2.23) is given by:

$$\begin{aligned} y(n) = & v^{(0)} + \sum_{\tau_1=1}^{D} v_{\tau_1}^{(1)} \cdot x_{\tau_1} + \sum_{\tau_1=1}^{D} \sum_{\tau_2=1}^{D} v_{\tau_1, \tau_2}^{(2)} \cdot x_{\tau_1} \cdot x_{\tau_2} + \\ & \cdots + \sum_{\tau_1=1}^{D} \cdots \sum_{\tau_p=1}^{D} v_{\tau_1, \ldots, \tau_p}^{(p)} \cdot x_{\tau_1} \cdot \cdots \cdot x_{\tau_p} + \cdots \end{aligned} \tag{2.24}$$

Where a bias has been added in the form of a Volterra kernel of order zero, $v^{(0)}$ .The functional $V^{(p)}\left[x_{\tau_1}, \cdots, x_{\tau_p}\right] = v^{(p)}_{\tau_1, \cdots, \tau_p} \cdot x_{\tau_1} \cdot \ldots \cdot x_{\tau_p}$ is p-linear, in the sense that:

$$V^{(p)}\left[c_{\tau_1} \cdot x_{\tau_1}, \cdots, c_{\tau_p} \cdot x_{\tau_p}\right] = c_{\tau_1} \cdot \ldots \cdot c_{\tau_p} V^{(p)}\left[x_{\tau_1}, \cdots, x_{\tau_p}\right]$$

$$V^{(p)}\left[\sum_{j_1} x_{j_1}, \cdots, \sum_{j_p} x_{j_p}\right] = \sum_{j_1} \cdots \sum_{j_p} V^{(p)}\left[x_{j_1}, \cdots, x_{j_p}\right] \tag{2.25}$$

And it can be demonstrated (Schetzen, 1989), by applying the commutative property in (2.24), that the Volterra kernel is symmetric:

$$v^{(p)}_{\tau_1,\ldots,\tau_p} = v^{(p)}_{perm(\tau_1,\ldots,\tau_p)} \tag{2.26}$$

where $perm(\tau_1,\ldots,\tau_p)$ is any possible permutation of the indexes $\tau_1,\ldots,\tau_p$.

The Volterra series can be regarded as Taylor series with memory. In fact, Volterra series share the same constraints on convergence. The systems modelled by Volterra series must yield an input-output map that is everywhere differentiable to an order of differentiation equal to the order of the series. For this reason, Volterra series can not be applied to systems with abrupt (i.e. discontinuous) changes in the output, e.g. bi-stable elements. This limitation however is not critical if the model is allowed to have a sufficient amount of *smoothness* in its mapping.

Volterra series have been studied in the context of *High-order Spectra* analysis of non-linear systems. (Schetzen, 1980, 1981; Nikias, Petropolou, 1993, Collis, 1996). Volterra series can exploit more complex relationships between inputs. It is possible to show that the higher-order polynomials in (2.24) are related to the higher-order moments of the input distribution. Higher-order analysis proves to be useful when dealing with non-linear and non-Gaussian model identification problems. The main

condition is for the system to be stationary, so that the kernels in (2.24) do not change. Different methods have been proposed to estimate the Volterra kernels in the frequency domain for continuous-time systems.

Collis *et al.* (1997) have considered the application of Volterra series to model a de-interlacing system. The motivation is that real images rarely possess Gaussian statistics. The results of this method are very encouraging, since it is possible to produce a significant increase in the performance in terms of MSE and in terms of subjective measures. Hence we will present the application of this technique in chapter 5, together with standard linear filtering, in order to provide a baseline performance comparison for the main subject of this work, the application of Radial Basis Function Networks. The two techniques will be compared in terms of qualitative performance (interpolation error) and complexity, i.e. computational load. Other aspects of Volterra series will be discussed in chapter 5 and in appendix B. A more complete study of Volterra series in de-interlacing is outside the scope of this thesis.

2.6.1. MMSE solution

In order to calculate the kernels one can exploit the single-layer structure of the Volterra series, as shown in figure 4 for $p=3$. In this context, the elements of Volterra kernels play the role of linear weights, while the multiplicative combinations of input elements play the role of node activation functions. Being the nodes non-parameterised, the Volterra series is completely linear-in-the-parameters. Hence Volterra kernels can be estimated in the MMSE sense by equation (2.21). We can form the intermediate vector

$$\mathbf{h} = \left[1\ x_1\ \ldots\ x_D\ x_1^2\ x_1x_2\ \ldots\ x_1x_D\ x_2^2\ x_2x_3\ \ldots\ x_2x_D\ \ldots\ x_1^p\ \ldots\right]^T = \tag{2.27}$$

$$= \left[\mathbf{h}^{(0)T}\ \mathbf{h}^{(1)T}\ \mathbf{h}^{(2)T}\ \ldots\ \mathbf{h}^{(p)T}\ \ldots\right]^T$$

where we have divided the intermediate vector in sub-elements according to the degree of the monomials in **h**.

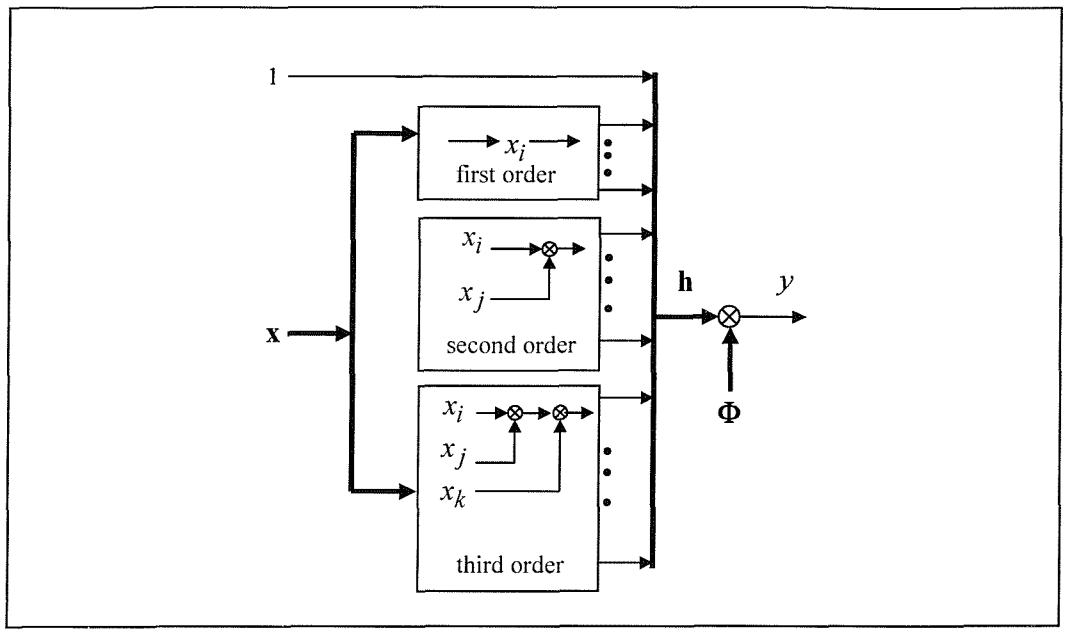

Figure 4. Single-layer representation of a third-order Volterra series.

Note how we have used the commutative property in (2.27) since the second-order products for the input x_2 starts from x_2^2 rather than x_2x_1. In other words, the term x_2x_1 is omitted because there is a x_1x_2 term that has previously been accounted for. From (2.27), The matrix **H** will assume the form:

$$\mathbf{H} = [\mathbf{h}_0 \ \cdots \ \mathbf{h}_{L-1}]^T$$

$$\mathbf{h}_n = \left[\mathbf{h}_n^{(0)T} \ \mathbf{h}_n^{(1)T} \ \mathbf{h}_n^{(2)T} \ \cdots \ \mathbf{h}_n^{(p)T} \ \cdots\right]^T \tag{2.28}$$

The corresponding kernel vector (i.e. weight vector $\mathbf{\Phi}$) will be:

$$\mathbf{\Phi} = \left[v^{(0)} \ v_1^{(1)} \ \cdots \ v_D^{(1)} \ v_{1,1}^{(2)} \ v_{1,2}^{(2)} \ \cdots \ v_{1,D}^{(2)} \ v_{2,2}^{(2)} \ v_{2,3}^{(2)} \ \cdots \ v_{1,\cdots,1}^{(p)} \ \cdots\right]^T \tag{2.29}$$

And the MMSE solution is given by (2.21). It is interesting to examine the structure of the matrices **R** and **P**. We limit our analysis to a third-order series, but the results can be extended to an arbitrary order. By using (2.28) we get:

$$\mathbf{H}^T\mathbf{H} = \begin{bmatrix} \sum_{n=0}^{L-1} \mathbf{h}_n^{(0)} \cdot \mathbf{h}_n^{(0)T} & \sum_{n=0}^{L-1} \mathbf{h}_n^{(0)} \cdot \mathbf{h}_n^{(1)T} & \sum_{n=0}^{L-1} \mathbf{h}_n^{(0)} \cdot \mathbf{h}_n^{(2)T} & \sum_{n=0}^{L-1} \mathbf{h}_n^{(0)} \cdot \mathbf{h}_n^{(3)T} \\ \sum_{n=0}^{L-1} \mathbf{h}_n^{(1)} \cdot \mathbf{h}_n^{(0)T} & \sum_{n=0}^{L-1} \mathbf{h}_n^{(1)} \cdot \mathbf{h}_n^{(1)T} & \sum_{n=0}^{L-1} \mathbf{h}_n^{(1)} \cdot \mathbf{h}_n^{(2)T} & \sum_{n=0}^{L-1} \mathbf{h}_n^{(1)} \cdot \mathbf{h}_n^{(3)T} \\ \sum_{n=0}^{L-1} \mathbf{h}_n^{(2)} \cdot \mathbf{h}_n^{(0)T} & \sum_{n=0}^{L-1} \mathbf{h}_n^{(2)} \cdot \mathbf{h}_n^{(1)T} & \sum_{n=0}^{L-1} \mathbf{h}_n^{(2)} \cdot \mathbf{h}_n^{(2)T} & \sum_{n=0}^{L-1} \mathbf{h}_n^{(2)} \cdot \mathbf{h}_n^{(3)T} \\ \sum_{n=0}^{L-1} \mathbf{h}_n^{(3)} \cdot \mathbf{h}_n^{(0)T} & \sum_{n=0}^{L-1} \mathbf{h}_n^{(3)} \cdot \mathbf{h}_n^{(1)T} & \sum_{n=0}^{L-1} \mathbf{h}_n^{(3)} \cdot \mathbf{h}_n^{(2)T} & \sum_{n=0}^{L-1} \mathbf{h}_n^{(3)} \cdot \mathbf{h}_n^{(3)T} \end{bmatrix} \tag{2.30}$$

Given the structure of the vectors $\mathbf{h}_n^{(0)}$, $\mathbf{h}_n^{(1)}$, $\mathbf{h}_n^{(2)}$ and $\mathbf{h}_n^{(3)}$, each element $\mathbf{h}_n^{(\alpha)} \cdot \mathbf{h}_n^{(\beta)T}$ in (2.27) is a matrix composed of monomials $x_{in}^{\alpha} \cdot x_{in}^{\beta}$. A similar consideration can be made on the elements of the vector **P**:

$$\mathbf{P} = \mathbf{H}^T\mathbf{Y} = \begin{bmatrix} \sum_{n=0}^{L-1} \mathbf{h}_n^{(0)} \cdot y_n & \sum_{n=0}^{L-1} \mathbf{h}_n^{(1)} \cdot y_n & \sum_{n=0}^{L-1} \mathbf{h}_n^{(2)} \cdot y_n & \sum_{n=0}^{L-1} \mathbf{h}_n^{(3)} \cdot y_n \end{bmatrix}^T \tag{2.31}$$

The structure of **R** and **P** in terms of the degree of their elements is depicted in figure 5. The elements of **R** can be seen as the estimates of higher-order moments of **x**, and the elements of **P** as the estimates of higher-order joint moments of **x** and y. It has been shown (Nikias, Petropolou, 1993) that if the joint density of the elements of **x** is symmetric with zero mean, then the joint moments of an odd number of elements are null. Therefore if **x** and y have zero-mean and symmetric joint distribution, then the odd-degree elements in **R** and **P** will be null.

Volterra series are completely linear in the parameters. However, the number of branches, *M*, is not a free parameter of the network, but is determined by the degree of the series, *p*, and by the dimension of the input *D*. These act as non-linear parameters that determine the complexity of the network. The number of branches in a Volterra series increases in proportion to D^p. This represents the main limitation to the application of Volterra series, because the computational load required can easily become unpractical (see table 1).

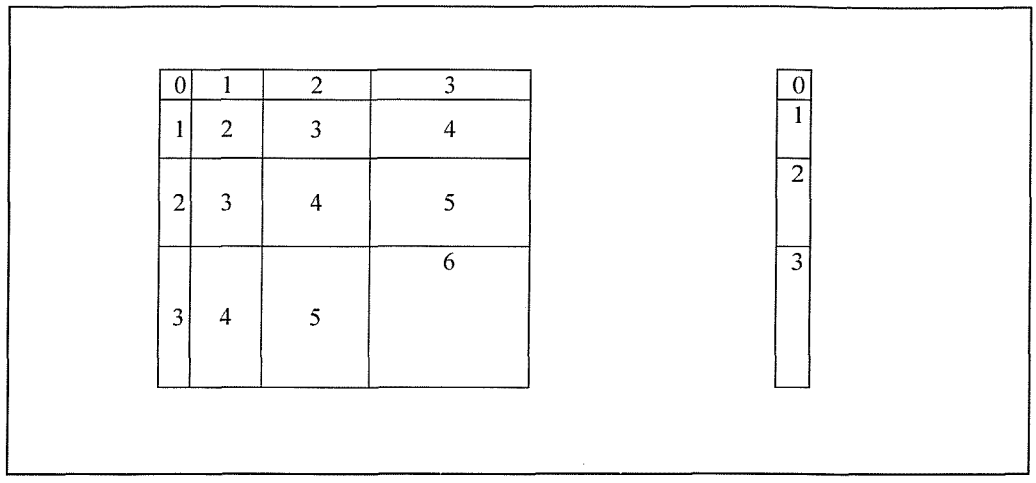

Figure 5. Degree of the elements in **R** (left) and **P** (right) for a third-order Volterra series.

		input dimension D				
		4	6	8	12	20
Degree p	1	4	6	8	12	20
	2	14	27	44	90	230
	3	34	73	164	454	1770
	4	69	199	494	1819	10625
	5	121	451	1286	6187	53129

Table 1. Number of nodes in Volterra series.

2.6.2. Skewness and Kurtosis.

In the study of higher-order moments, particular importance is given to the third and fourth-order central moments, $\mu_3 = E\left[(\mathbf{x} - E[\mathbf{x}])^3\right]$ and $\mu_4 = E\left[(\mathbf{x} - E[\mathbf{x}])^4\right]$. Two derived quantities are the *skewness* μ_3/σ^3 and *kurtosis* μ_4/σ^4. From the previous discussion, is clear that a symmetric distribution has zero skewness. Moreover, for a scalar Gaussian variable it can be shown that the kurtosis is equal to 3 (Nikias, Petropolou, 1993). Hence these two quantities assume an importance as indicators of non-Gaussianity. In figure 6, examples of probability density functions are shown, with non-zero skewnesses, and with kurtoses different from 3.

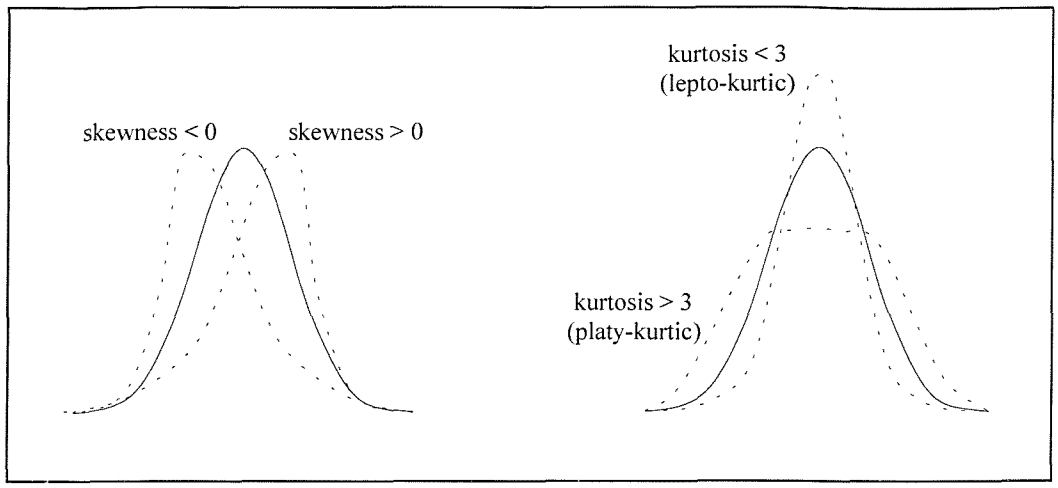

Figure 6. Examples of skewed and kurtic density functions (dashed line), compared to Gaussian functions (solid line).

2.7. Radial Basis Function Networks

The main task of this thesis is the study of a particular single-layer network called Radial Basis Function Network (RBFN) (Powell, 1987; Broomhead, Lowe 1988). RBFNs are able to generate arbitrary multi-dimensional mappings, and have a strong statistical background. RBFNs will be discussed in depth in the next chapter.

Many works illustrate the application of RBFN techniques to image processing. Examples are given by Sherstinsky and Picard (1992), Rosenblum *et al.* (1994), Arad and Reisfeld (1995), Howell and Buxton (1995), Kassaam and Cha (1996), Gutta and Wechsler (1996).

2.8. Multi-layer networks

In the previous sections we have studied the main properties of single-layer networks. In order to achieve a better understanding of these structures, in this section we will briefly discuss multi-layer networks. A multi-layer network with p layers is a structure described by the equation

$$y = \sum_{j_p=0}^{M_p} \phi_{j_p} h_{j_p}^{(p)} \left(\sum_{j_{p-1}=0}^{M_{p-1}} \phi_{j_{p-1}} h_{j_{p-1}}^{(p)} \left(\dots \sum_{j_1=0}^{M_1} \phi_{j_1} h_{j_1}^{(1)} \left(\boldsymbol{\Phi}_{j_1}^T \mathbf{x} + \phi_{j_1} \right) \right) \right) \quad (2.32)$$

Note that in (2.32) the activation functions are fed with a linear combination of the input. This linear operation is part of the activation function and it is different for each node. It is not be considered as a layer, but rather as part of the $R^D \rightarrow R$ mapping performed by each h_{j_1}. A common choice for the non-linear activation functions is often the *sigmoidal* function

$$h(r) = \frac{1}{1 + exp(-\sigma \cdot r)} \tag{2.33}$$

If $\sigma \rightarrow \infty$, equation (2.29) becomes the *threshold activation function*

$$h_{j_p}^{(p)}(r) = \begin{cases} 1 & r \geq 0 \\ 0 & r < 0 \end{cases} \tag{2.34}$$

The most studied class of multi-layer network is the two-layer network. Kolmogorov's theorem (Kolmogorov, 1957; Lorentz, 1976) states that a network with two layers of univariate non-linear activation functions can exactly interpolate any multi-variate continuous function. Unfortunately, it has been also shown that the activation functions are highly non-smooth (Vitushkin, Henkin 1977), and as complex as the original functions in terms of computational complexity.

However, multi-layer networks prove to be effective in many applications, especially in classification and decision problems. These networks typically require a significant amount of non-linear training in order to find the set of weights ϕ_{j_p}, the so-called *backprogagation* algorithm (Rumelhart *et al.*, 1986) is used to relate the output error to the different layers of weights. Multi-layer networks are not the subject of this thesis, so we will not continue with their description.

2.9. Conclusions

This chapter has presented the general issues involved in the design of an interpolation system based on the single-layer feed-forward architecture. The principle of training a flexible (i.e. parametrised) system in order to approximate an observed input-output set (learning by examples) has been explained in its most relevant aspects.

Firstly, it has been shown how the complexity of the model is related to the size of the training set. As the latter increases, more complexity is in principle required to account for the new information available. However this approach not only leads to unpractical solutions, but it also fails one basic goal of a learning algorithm, which is the ability to predict patterns not seen during the training process. This ability is usually referred as generalisation. Chapter 7 will examine this issue in much more detail.

An important problem to tackle is the choice of the cost function that measures the performance of the model. An analysis has been carried on a particular cost function, the minimum mean square error (MSE). This function is a popular choice because it usually leads to a closed solution for the optimal parameters. This is particularly true for the linear weights of a single-layer network trained in the MMSE sense. However, other choices are available, that might better exploit the peculiar characteristics of the system at the price of a more expensive training.

The choice of the functional mapping yield by the network is fundamental in order to model the complexity of the system with the minimum computational cost. More importantly it is necessary to avoid excessive specialisation on the particular training set, in order to produce a general result. As the complexity of the model increases, it is easier for the training algorithm to over-fit the data set and reduce the generality of the mapping.

Volterra series is an intuitive extension of the linear model and, although not the main focus of this thesis, a general survey is presented in this chapter. By introducing higher-order monomial terms it is possible to account for more complex dependencies

between the input pixels. Chapter 5 will show how this richer set of relationships will lead to the expected performance gains. The degree of the series and the number of taps in the sampling lattice determine the non-linear structure of the network. Therefore Volterra series are completely linear in the parameters and MMSE training leads to a closed solution for the optimal parameters. The major drawback of Volterra series is that seeking for more complexity by increasing the sampling aperture or the degree the series leads to unpractical computational costs. This is particularly the case when modelling steep variations in the mapping.

Radial Basis Function Networks represent the main focus of this thesis and the next chapter will be dedicated to the subject. Multi-layer networks are briefly introduced for the sole purpose of a better understanding of the single-layer architecture. However their application to the de-interlacing problem is not considered in this thesis.

3. RADIAL BASIS FUNCTION NETWORKS

3.1. Introduction

In the previous chapter we have described the main features of a family of techniques known as single-layer networks. We have seen how linear and polynomial series can be modelled as particular cases of this structure. In this chapter we discuss a single-layer architecture where each node's activation function is a function of the Euclidean distance between the input vector and a prototype vector called the *centre*. Because of the radial symmetry of the distance measure in the input space, these networks are called Radial Basis Function Networks (RBFN) (Powell, 1987; Broomhead, Lowe 1988). RBFN are able to create arbitrary mappings by a superposition of these radial functions. The basic philosophy behind this technique is that each node acts in a local fashion, i.e. deals with a particular zone of the input space. The centre of symmetry of each node's activation function defines the middle of this zone.

Like linear filters and Volterra series, RBFN form a unifying link between different concepts such as interpolation, regularisation and density estimation. The consequence of such an interaction is that we can devise efficient training procedures that draw their justification from different theoretical backgrounds. RBFN are different from linear filters and Volterra series in that centres act as non-linear parameters.

One basic training principle is that parameters governing inner units can be trained using either *supervised* or *unsupervised* procedures (i.e., using the input set **X** respectively with or without an association with a corresponding target set **T**). Unsupervised training uses the non-linear parameters to estimate the input distribution, leaving the linear parameters to map the input-output dynamics. Supervised training uses both non-linear and linear parameters to estimate the mapping. We will investigate this point shortly. Many works illustrate the application of RBFN techniques in non-linear system identification and in image processing. Examples are given by Chen *et al.*, (1990), Sherstinsky and Picard (1992), Rosenblum

et al. (1994), Arad and Reisfeld (1995), Howell and Buxton (1995), Kassaam and Cha (1996), Gutta and Wechsler (1996).

3.2. RBFN Architecture

The general form of a single-output RBFN is (Broomhead, Lowe 1988):

$$y_n = \phi_0 + \sum_{j=1}^{M} \phi_j \, h_j(\mathbf{x}_n) = \phi_0 + \sum_{j=1}^{M} \phi_j \, h(\| \mathbf{x}_n - \mathbf{c}_j \|; \sigma_j) \qquad (3.1)$$

The first function applied to the input is the Euclidean distance $r_j = \|\mathbf{x}_n - \mathbf{c}_j\|$ of the current input $\mathbf{x}_n$ from a vector $\mathbf{c}_j \in R^D$ called the j-th centre of the RBFN. The resulting value will be passed to the function $h(r_j; \sigma_j): R^+ \rightarrow R$. The support of h in R^+ is controlled by the *width* parameter σ_j. Note that the presence of this parameter is not necessary to identify a structure as RBFN. The important point is that each node realises a mapping $R^D \rightarrow R$ that has radial support in R^D. The single-layer structure of the network is evident.

3.3. Exact interpolation

Historically, Radial Basis Function methods originated as methods to perform exact interpolation of a multi-dimensional function (Powell, 1987). The exact interpolation problem requires each multi-dimensional input to be exactly mapped into the corresponding target. Consider a training set of M input-output patterns. Our goal is to find a function $y = f(\mathbf{x})$ such that

$$y_j = f(\mathbf{x}_j) = t_j, \quad \forall j = 1 \ldots M \qquad (3.2)$$

The Radial Basis Function approach introduces a set of M basis functions, one for each data point, which take the form of $h(\| \mathbf{x} - \mathbf{x}_j \|)$ where $h(r), r = \| \mathbf{x} - \mathbf{x}_j \|$ is

some form of non-linear function which will be discussed shortly. The output mapping is given by a weighted sum of these basis functions

$$y(\mathbf{x}) = \sum_{j=1}^{M} \phi_j \, h(\| \mathbf{x} - \mathbf{x}_j \|) \tag{3.3}$$

Clearly, the interpolating function must be sufficiently smooth in order to generate a realistic map, that correctly estimates the output corresponding to inputs outside the training set. For example the activation function $h(\| \mathbf{x} - \mathbf{x}_j \|) = \delta(\| \mathbf{x} - \mathbf{x}_j \|)$ achieves perfect interpolation in $\mathbf{x}_j$, but is null elsewhere. Using the training set $\{\mathbf{x}_j, t_j\}$ it is possible to write equation (3.3) in matrix form:

$$\mathbf{H}\Phi = \mathbf{Y} \tag{3.4}$$

where $\mathbf{H}$ is a square matrix such that $H_{ij} = h(\| \mathbf{x}_i - \mathbf{x}_j \|)$. The system (3.4) is easily solved:

$$\Phi = \mathbf{H}^{-1} \mathbf{Y} \tag{3.5}$$

provided $\mathbf{H}$ is invertible. It has been shown (Schoenberg, 1938; Micchelli, 1986) that for a large class of functions h the matrix $\mathbf{H}$ is invertible. Examples of this class are the Gaussian function

$$h(r;\sigma) = exp(-r^2/\sigma^2) \tag{3.6a}$$

the multi-quadratic function

$$h(r;\sigma) = \sqrt{r^2 + \sigma^2} \tag{3.6b}$$

and the linear function

$$h(r;\sigma)=\sigma \cdot r \tag{3.6c}$$

It is important to notice that these are sufficient conditions for the invertibility of **H**. For instance, the popular *thin plate spline function* (Powell 1987)

$$h(r)=r^2\, ln(r) \tag{3.6d}$$

does not satisfy Micchelli's conditions (Poggio, Girosi 1989). Nevertheless its effectiveness has been demonstrated in many applications. It has been shown (Powell, 1987) that in the context of exact interpolation the exact analytical expression for h is not critical. This empirical statement eases the problem of finding the optimal non-linear parameter set. In the experiments however, it will be shown that this assumption must be considered with great care, since it will be shown to be only valid when the number M of nodes is high. Furthermore, it will be shown that the width parameter has a significant effect on the resulting error. As a final remark, note that in the context of exact interpolation the centres are not considered as parameters, since they are determined once and for all by the training set. Furthermore, in the Schoenberg's solution for the invertibility problem, the activation functions shown in equations (3.6) are non-parametric, i.e. without a width parameter. Hence, in their original formulation, RBFN are purely linear in the parameters.

In this work we will focus on the Gaussian function (3.6a) (Gaussian RBFN, GRBFN). This function presents, together with an intuitive localisation property, some interesting analytical properties and an immediate link with the theory of density estimation using *Gaussian Mixtures* (Titterington et al., 1985). These properties will be discussed later in this chapter.

3.4. Radial Basis Function Networks

The Radial Basis Function interpolator suffers from a number of serious limitations. One major drawback is the number of basis functions required, which is given by the number of input-output patterns. This condition often leads to impracticably large networks.

Another limitation, possibly more important, is that exact interpolation gives very poor generalisation. In other words, a network built to exactly interpolate a specific pattern generally exhibits a severe oscillating behaviour away from the interpolated points, especially in presence of noisy data (overfitting). It is important to remember that we want to generate a model able to describe the general behaviour of the data rather than an exact match of the training set. It has already been discussed the need for a smooth interpolation of data, possibly loosing the exact interpolation constraints.

By introducing a number of modifications to the exact interpolation procedure we obtain the Radial Basis Function Network model (Broomhead and Lowe, 1988; Moody and Darken, 1989). These modifications are:

- The number M of basis functions is commonly much smaller than the number of training patterns L.

- The centres for the basis functions are no longer constrained to be picked from the data, but can be moved in the input space during the training process.

- A bias parameter is introduced to account for the difference between the average value of basis functions over the input set and the average value of the output.

The aim is to yield a smooth interpolation with the number of basis functions determined by the complexity of the mapping rather than by the size of the training set. It has been shown that RBFN is capable of *universal approximation* (Girosi and Poggio, 1989b; Hartman *et al.* 1990); i.e. a RBFN with a suitable number of basis function can approximate any mapping. Clearly, the promotion of the centres to the

rank of free parameters raises the problem of finding their optimal values. Like width parameters, the problem of finding optimal centres is non-linear.

3.5. Extensions of the RBFN model

The model proposed in (3.1), with Gaussian basis functions, can be extended to allow *generalised covariance matrices* $\mathbf{\Sigma}_j$ (Musavi *et al.*, 1992) for each basis function j (Multi-variate GRBFN).

$$y(\mathbf{x}) = \sum_{j=1}^{M} \phi_j \, exp\left[-\left(\mathbf{x}-\mathbf{c}_j\right)^T \mathbf{\Sigma}_j^{-1} \left(\mathbf{x}-\mathbf{c}_j\right)\right] \tag{3.7}$$

Where $\Delta_j^2 = \left(\mathbf{x}-\mathbf{c}_j\right)^T \mathbf{\Sigma}_j^{-1} \left(\mathbf{x}-\mathbf{c}_j\right)$ is called the *Mahalanobis distance* (squared) between $\mathbf{x}$ and $\mathbf{c}_j$. The immediate advantage of multivariate Gaussians is that they have a more general (elliptical) support in the input space, possibly requiring fewer basis functions than the circular case. Obviously, this approach leads to a more complicated, and possibly more costly, architecture, as the number of parameters is increased. Examples of multivariate GRBFN can be found in Musavi et al. (1992), Kassaam and Cha (1993).

In a Hybrid GRBFN (Kassaam and Cha 1993) an explicit linear path is added in parallel to the GRBFN to give:

$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + \sum_{j=1}^{M} \phi_j \, exp\left[-\left(\mathbf{x}-\mathbf{c}_j\right)^T \mathbf{\Sigma}_j^{-1} \left(\mathbf{x}-\mathbf{c}_j\right)\right] \tag{3.8}$$

where $\mathbf{w}$ is a vector of linear weights. This model is very important in terms of network efficiency vs. complexity (i.e. size), whenever the interpolation problem can be divided into a linear and non-linear part. It has also been suggested (Girosi and Poggio, 1990) that convergence of non-linear methods is improved by the addition of a linear path.

Other extensions found in literature are the *normalised GRBFN* (Kassaam and Cha 1993):

$$y(\mathbf{x}) = \sum_{j=1}^{M} \frac{\phi_j \, exp\left[-\left(\mathbf{x}-\mathbf{c}_j\right)^T \boldsymbol{\Sigma}_j^{-1} \left(\mathbf{x}-\mathbf{c}_j\right)\right]}{\sum_{k=1}^{M} \phi_k \, exp\left[-\left(\mathbf{x}-\mathbf{c}_k\right)^T \boldsymbol{\Sigma}_k^{-1} \left(\mathbf{x}-\mathbf{c}_k\right)\right]} \tag{3.9}$$

and the connectionist normalised local spline (CNLS) (Jones et.al. 1990):

$$y(\mathbf{x}) = \sum_{j=1}^{M} \left[w_j + \mathbf{G}_j \left(\mathbf{x}-\mathbf{c}_j\right)^T\right] \cdot \frac{\phi_j \, exp\left[-\left(\mathbf{x}-\mathbf{c}_j\right)^T \boldsymbol{\Sigma}_j^{-1} \left(\mathbf{x}-\mathbf{c}_j\right)\right]}{\sum_{k=1}^{M} \phi_k \, exp\left[-\left(\mathbf{x}-\mathbf{c}_k\right)^T \boldsymbol{\Sigma}_k^{-1} \left(\mathbf{x}-\mathbf{c}_k\right)\right]} \tag{3.10}$$

Where $\mathbf{G}_j$ is an additional vector of parameters. The terms $\mathbf{G}_j\left(\mathbf{x}-\mathbf{c}_j\right)^T$, coupled with normalisation, provide a means to utilise information on the gradient of the desired function in proximity of each centre. This extension will be discussed shortly in the context of Gaussian Mixtures models. Clearly the models (3.8), (3.9) and (3.10) can be also derived for the mono-variate model (3.1).

In Girosi and Poggio (1989), RBFN are studied as part of a more complex model given by:

$$y(\mathbf{x}) = \sum_{i=1}^{M} \phi_j \, h\left(\left\|\mathbf{x}-\mathbf{x}_j\right\|\right) + \sum_{s=1}^{S} d_s \, p_s(\mathbf{x}) \tag{3.11}$$

where $\{p_s(\mathbf{x}) \,|\, s=1\ldots S\}$ is a basis of $\pi_{k-1}\left(R^D\right)$, the space of polynomials $R^D \rightarrow R$ with degree at most *k-1*. The model (3.10) is a polynomial generalisation of the hybrid model (3.8).

3.6. Training of RBFN

A key aspect of RBFN training is the difference between linear and non-linear parameters. The theoretical link between RBFN and density estimation (section 3.7), together with the space mapping nature of the network suggests that the centres could be trained with an unsupervised algorithm (Musavi *et al.*, 1992; Bishop, 1995; Fung, *et al.*, 1996). The goal of this phase should be some form of density mapping of the input space, without considering the relationship with the target data (unsupervised training). In other words, centres should be placed in a way to resemble the distribution of the inputs in the input space, regardless the associated dynamics of the target data. The second (linear) layer is successively trained with supervised linear techniques (MMSE estimate) to match the estimated input distribution with the target data. However, classic supervised optimisation methods like *simplex* or derivative-based algorithms work equally well.

Given a training set $\{\mathbf{X},\mathbf{T}\}$ of L examples and M basis functions, it is possible to derive a MMSE solution for $\mathbf{\Phi}$ from (3.4)

$$\mathbf{\Phi}^* = \mathbf{H}^+\mathbf{T} \tag{3.12}$$

where $\mathbf{H}^+$ is the pseudo-inverse of $\mathbf{H}$ introduced in section 2.5. The problem remains to find the MMSE solution for $\mathbf{H}$, i.e. to find a good minimiser for first layer's free parameters.

If we consider the training of a Gaussian RBFN, each node of the first layer has two parameters, the centre and width. The simplest approach is to select the M centres from a uniform sampling of the input space. Clearly the idea is to make a "copy" of the input distribution, and as such is an unsupervised approach. The assumption that a random sampling of the input set generates a faithful, reduced copy of the input's distribution is generally correct, if a large number of centres is chosen. However, this approach may lead to impractical, large networks.

In general, RBFN are affected by the so-called "*curse of dimensionality*" from the very nature of the way they create maps. This problem is particularly severe when noisy patterns are presented to the network. These inputs obviously increase the dimensionality of the problem, but in fact represent a nuisance factor in determining the intrinsic dimensionality of the model.

It has been shown (Barron, 1993) that the residual MSE falls as M^{-1}, irrespective of the dimensionality D of the input space. Conversely, a polynomial model with M coefficients generally decays as $M^{-2/D}$. Thus RBFN offer an advantage whenever we deal with high dimensional problems. The number of nodes can be reduced when the basis functions include a covariance matrix $\mathbf{\Sigma}$. However, this significantly increases the number of parameters required to define a node.

Another difficulty arises from the task of choosing widths σ_j. In principle, we require the width parameter to create a "smooth" mapping with high accuracy. Intuitively, large values of σ_j (broad Gaussians) make the mapping very smooth. Conversely, small values of σ_j (narrow Gaussians) are able to match the "fine detail" in the output dynamics.

The problem of determining optimal widths is in principle non-linear. However, to trade-off narrow Gaussians against broad Gaussians, we can consider some measure of input space density around each centre $\mathbf{c}_j$. The approach developed in this work is to calculate the mean distance $\overline{\delta}_j$ of the other centres from $\mathbf{c}_j$, and determine σ_j ensuring that the Gaussian function $h_j(r)$ has a pre-determined value at the mean distance $\overline{\delta}_j$. This pre-determined value controls the trade off between smoothness and accuracy. A similar technique has been applied to multivariate Gaussians by Musavi *et al.*, (1992). This approach will be more completely described in chapter 6. It will be also shown that the use of hybrid GRBFNs simplifies the problem of determining suitable width parameters.

A more general approach necessarily involves some form of non-linear optimisation based on iterative search techniques. Fletcher (1987) gives a comprehensive review of non-linear minimisation techniques. Nelder and Mead (1965) proposed an optimisation algorithm based on modified *simplex* technique. This algorithm is more completely discussed in appendix C and it will be applied in chapter 6.

3.7. GRBFN and Gaussian Mixtures

RBFN have a very strong bond with the field of density estimation. One density model, the *mixture distribution* (Titterington *et al.*,1985; McLahan and Basford, 1988) makes the assumption that a stochastic process is the ensemble of M stochastic processes whose distributions have a simple parametrised functional form that is localised in space.

In terms of density function, we can then write our global function $p(\mathbf{x})$ as a linear combination of class conditional densities $p(\mathbf{x}|j)$

$$p(\mathbf{x}) = \sum_{j=1}^{M} p(j) \cdot p(\mathbf{x}|j) \tag{3.13}$$

$p(j)$ is the *prior* probability of class j. These priors satisfy the constraints

$$\sum_{j=1}^{M} p(j) = 1 \tag{3.14a}$$

$$0 \le p(j) \le 1 \tag{3.14b}$$

Using Bayes's theorem, we can determine the *posterior* probability of component j having generated $\mathbf{x}$

$$p(j|\mathbf{x}) = \frac{p(\mathbf{x}|j)p(j)}{p(\mathbf{x})} \tag{3.15}$$

These posteriors, as functions of j, are the likelihood function of class j given the observed data **x**. The Gaussian mixture model is a mixture distribution with Gaussian class conditional densities

$$p(\mathbf{x}|j) = \frac{1}{(2\pi)^{D/2} |\boldsymbol{\Sigma}_j|^{1/2}} \cdot \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)\right\} \qquad (3.16)$$

Maximum Likelihood methods like *Expectation-maximisation* (EM) algorithm (Dempster *et. al.*, 1997) have been applied to optimise Gaussian mixtures. The EM algorithm is an iterative parameter updating procedure that combines Bayes's theorem together with gradient calculation to compute the maximum likelihood estimate of the parameters given the current training input and the previous parameter estimates. A review is given by Redner and Walker (1994).

The formal similarity between Gaussian mixtures and GRBFN is evident. One should note that Gaussian mixtures only model the input distribution. There is no link to the output. In fact, the formal equivalence between the prior $p(j)$ and the ϕ_j parameter in RBFN is misleading. Priors $p(j)$ model the relative influence of different classes in order to describe a density function of a variable **x** in R^D. In RBFN ϕ_j models the relative influence of different nodes in order to interpolate a function $R^D \rightarrow R$. An example of this difference is depicted in figure 1 for the *1*-dimensional case.

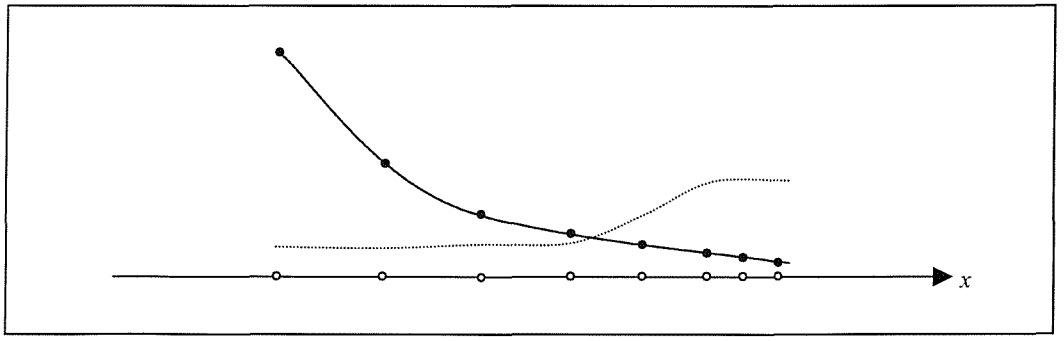

Figure 1. Gaussian Mixtures model the density function (dashed line) of the unlabeled data (white dots on the axis). RBFN mapping (solid line) attempts to interpolate the targets (black dots) associated with the data. In the example the target function is greater on the left side of the axis, while the input density is greater on the right side of the axis.

A possible solution is to discard the priors and calculate ϕ_j with standard linear methods. A more generalised approach is given by the theory of *probabilistic neural networks*, PNN (Perlovsky and McManus, 1991; Streit and Luginbuhl, 1994). An intersting derivation for a MMSE interpolator based on PNN is given by Cha and Kassaam (1995).

They model both input and output as realisations of two correlated processes **X** and **Y** described by a mixture of M Gaussian vectors $\mathbf{Z}_j = [\mathbf{X}_j \ \mathbf{Y}_j]^T$ in R^{D+1}. The prior of each population j is $p(j) = \lambda_j$, whilst its mean and covariance matrix are given by:

$$m_j = \begin{pmatrix} \mathbf{c}_j \\ w_j \end{pmatrix} \qquad \boldsymbol{\Sigma}_j = \begin{bmatrix} \boldsymbol{\Sigma}_{\mathbf{x}_j} & \boldsymbol{\Sigma}_{\mathbf{x}_j y_j} \\ \boldsymbol{\Sigma}_{\mathbf{x}_j y_j} & \boldsymbol{\Sigma}_{y_j} \end{bmatrix} \tag{3.17}$$

Where $\mathbf{c}_j = E[\mathbf{X}_j]$, $w_j = E[\mathbf{Y}_j]$, and:

$$\begin{aligned} \boldsymbol{\Sigma}_{\mathbf{x}_j} &= E[\mathbf{X}_j \mathbf{X}_j^T] - \mathbf{c}_j \mathbf{c}_j^T \\ \boldsymbol{\Sigma}_{y_j} &= E[\mathbf{Y}_j \mathbf{Y}_j^T] - w_j^2 \\ \boldsymbol{\Sigma}_{\mathbf{x}_j y_j} &= \boldsymbol{\Sigma}_{y_j \mathbf{x}_j}^T = E[\mathbf{X}_j \mathbf{Y}_j^T] - \mathbf{c}_j w_j \end{aligned} \tag{3.18}$$

From these assumptions Cha and Kassaam demonstrate that the MMSE estimate of y is given by:

$$y_{MMSE}(\mathbf{x}) = \sum_{j=1}^{M}\left[w_j + \Sigma_{\mathbf{x}_j y_j} \Sigma_{\mathbf{x}_j}^{-1} (\mathbf{x} - \mathbf{c}_j) \right] \cdot$$

$$\cdot \frac{\lambda_j \left|\Sigma_{\mathbf{x}_j}\right|^{-\frac{1}{2}} \cdot exp\left[-\frac{1}{2} (\mathbf{x} - \mathbf{c}_j)^T \Sigma_{x_j}^{-1} (\mathbf{x} - \mathbf{c}_j) \right]}{\sum_{k=1}^{M} \lambda_k \left|\Sigma_{\mathbf{x}_k}\right|^{-\frac{1}{2}} \cdot exp\left[-\frac{1}{2} (\mathbf{x} - \mathbf{c}_k)^T \Sigma_{\mathbf{x}_k}^{-1} (\mathbf{x} - \mathbf{c}_k) \right]} \tag{3.19}$$

which has the formal structure of a CNLS. It is also suggested that the EM algorithm can be used to train this network.

3.8. GRBFN and Steepest Descent techniques

GRBFN can be trained using classic line-search non-linear optimisation methods based on first or second derivatives. Given the MSE expression:

$$C = \sum_{n=0}^{L-1} e_n^2 = \sum_{n=0}^{L-1} \left[t_n - \sum_{j=1}^{M} \phi_j exp\left(-\frac{\left\| \mathbf{x}_n - \mathbf{c}_j \right\|^2}{\sigma_j^2} \right) \right]^2 \tag{3.20}$$

it is straightforward to calculate the partial derivatives:

$$\frac{\partial C}{\partial \phi_j} = \sum_{n=0}^{L-1} e_n \, exp\left(-\frac{\left\| \mathbf{x}_n - \mathbf{c}_j \right\|^2}{\sigma_j^2} \right) \tag{3.21a}$$

$$\frac{\partial C}{\partial \mathbf{c}_j} = -2 \sum_{n=0}^{L-1} e_n \phi_j \frac{(\mathbf{x}_n - \mathbf{c}_j)}{\sigma_j^2} exp\left(-\frac{\left\| \mathbf{x}_n - \mathbf{c}_j \right\|^2}{\sigma_j^2} \right) \tag{3.21b}$$

$$\frac{\partial C}{\partial \sigma_j} = 2 \sum_{n=0}^{L-1} e_n \phi_j \frac{\left\| \mathbf{x}_n - \mathbf{c}_j \right\|^2}{\sigma_j^3} exp\left(-\frac{\left\| \mathbf{x}_n - \mathbf{c}_j \right\|^2}{\sigma_j^2} \right) \tag{3.21c}$$

Analogous expressions can be obtained for second partial derivatives and the *Hessian* matrix used in *Newton-like* methods (Fletcher, 1987). Notice that given two generic parameters, α_i, β_j, belonging to two different bases i, j

$$\frac{\partial^2 C}{\partial \alpha_i \partial \beta_j} = 0 \ \ if \ i \neq j \tag{3.22}$$

3.9. RBFN and Regularisation theory

RBFN possess a strong bound with the theory of *regularisation* (Tikhonov, 1963; Tikhonov, Arsenin, 1977; Morozov, 1984; Bertero, 1986; Poggio, Girosi, 1989; Girosi *et al.*, 1995). Regularisation is a way to control the smoothness of a mapping introducing terms in the cost function that penalise non-smooth maps. By doing so, we make sure that the mapping does not overfit the data, and has a degree of smoothness to deal with patterns that have not been included in the training set. Hence regularisation can be also viewed as a generalisation technique. A regularised MMSE training attempts to minimise the quantity

$$C = \sum_{n=0}^{L-1} [y(\mathbf{x}_n) - t_n]^2 + \nu \Omega \tag{3.23}$$

where Ω is the regularisation term that is assumed to increase as the mapping overfits the data, and ν is called the regularisation parameter. An example of regulariser for a one input, one output system is given by the class of Tikhonov regularisers (Tikhonov and Arsenin, 1977):

$$\Omega = \frac{1}{2} \sum_{r=0}^{R} \int_a^b \Psi_r(x) \left(\frac{d^r y}{dx^r} \right)^2 dx \tag{3.24}$$

Without discussing the details of the formulation, it is clear that the regularisation term is inversely proportional to the smoothness of the mapping, since the derivatives of $y(\mathbf{x})$ will increase as the mapping becomes less smooth.

Girosi and Poggio (1989) proposed a regularised cost function that assumes the general form

$$\Omega = \int |Q[y(\mathbf{x})]| d\mathbf{x} \tag{3.25}$$

where Q is some differential operator. Non-smooth mappings will give large values of $|Q[y(\mathbf{x})]|$ and hence will be penalised. This approach leads to a more general network mapping given by:

$$y(\mathbf{x}) = \sum_{j=1}^{M} \phi_j \cdot G\left(\left\| \mathbf{x} - \mathbf{x}_j \right\|\right) \tag{3.26}$$

Where G is the *Green's function* of the operator $\hat{Q}Q$ and is defined as:

$$\hat{Q}\{Q[G(\mathbf{x},\mathbf{x}')]\} = \delta(\mathbf{x} - \mathbf{x}') \tag{3.27}$$

$\hat{Q}$ is the *adjoint* differential operator to Q. If Q is rotationally invariant, then the Green's functions depend only on $\left\| \mathbf{x} - \mathbf{x}_j \right\|$, and so they are radial functions. If the differential operator Q is given by:

$$\int |Q[y(\mathbf{x})]|^2 d\mathbf{x} = \sum_{l=0}^{\infty} \frac{\sigma^{2l}}{l! 2^l} \int \left| D^l y(\mathbf{x}) \right|^2 d\mathbf{x} \tag{3.28}$$

where $D^{2l} = \left(\nabla^2\right)^l$, $D^{2l+1} = \nabla\left(\nabla^2\right)^l$ with ∇ and ∇^2 denoting the gradient and Laplacian respectively, then it is proved (Weiss, 1987; Yuille and Grzywacz, 1988; Girosi and Poggio 1989) that the Green's function is a Gaussian with width σ.

Bishop (1991b) gives another example of a regularisation term:

$$\Omega = \sum_{n=0}^{L-1} \sum_{i=1}^{D} \left(\frac{\partial y(\mathbf{x}_n)}{\partial x_i} \right)^2 \tag{3.29}$$

which penalises mappings with large curvature. The regularised MMSE solution $\mathbf{\Phi}^*$ can be found solving the system:

$$\mathbf{M\Phi} = \mathbf{H}^T \mathbf{T} \tag{3.30}$$

where

$$\mathbf{M}_{jk} = \sum_{n=0}^{L-1} \left\{ h_{j_n} h_{k_n} + \nu \sum_{i=1}^{D} \left[\frac{\partial^2 h_{j_n}}{\partial x_i^2} \cdot \frac{\partial^2 h_{k_n}}{\partial x_i^2} \right] \right\} \tag{3.31}$$

One of the simplest forms of regulariser is the *weight decay* regulariser (Hinton, 1987; Bishop, 1995)

$$\Omega = \sum_{i=1}^{M} \phi_i^2 = \mathbf{\Phi}^T \mathbf{\Phi} \tag{3.32}$$

The intuitive idea is that overfitted mappings usually yield large weights, while smooth mappings yield smaller weights. This technique will be more completely investigated in chapter 7.

3.10. Conclusions

This chapter illustrates the basic principles and training techniques of RBFN. Originally developed in the context of exact interpolation, RBFN are able to generate a smooth interpolation of a set of scattered input-output patterns, under a reasonable set of conditions. The main advantage of RBFN is that it is possible to devise simple training strategies for the network parameters. In theory RBFN require the determination of a set of non-linear parameters. However the principle that the node's activation function is "localised" on an area of the input space determined by the

centre allows one to devise simple training strategies that overcome the necessity of non-linear training. These linear training strategies produce satisfactory results if a sufficiently high number of centres (i.e. nodes) is allowed. More accurate results can be obtained for smaller networks if non-linear techniques are adopted.

The basic RBFN architecture can be easily extended to provide more efficient mappings in terms of the number of centres employed. An example is the hybrid architecture that implements an explicit linear path in the RBFN.

RBFN have two interesting theoretical connections. The first is with the theory of statistical density estimation, specifically with the Gaussian Mixtures model. It is important to notice that density estimation and functional approximation are in principle two different goals. However by considering the joint densities of the input-output pattern it is possible to put RBFN and GM into a single training framework as illustrated in section 3.7.

Another important point in favour of RBFN comes from the theory of regularisation. This has been developed as a general methodology for approximation in order to ensure a smooth approximating function. This goal is achieved by including regularisation terms in the error function that penalise non-smooth mappings. It is possible to show that Gaussian RBFN satisfy one possible choice for the regularisation function. Other interesting aspects of RBFN with regards to the regularisation theory will be illustrated in chapter 7, in the context of generalisation.

4. ORTHOGONAL LEAST SQUARES REDUCTION

4.1. Introduction

In chapter 2, we have discussed the necessity of finding suitable techniques in order to determine the optimal size of a network (number of nodes). Clearly, the optimal number of nodes is a trade-off between the accuracy of the solution, and the computational requirements. An accurate match to the training pattern is generally obtained at the cost of a large network. Conversely a smaller network, which may satisfy practical constraints, will realise a poorer fit of the data. One should remember that, aside computational considerations, there are two other main reasons to keep the network size small. Firstly, noisy data require large networks, in order to match the random fluctuations of the training set's dynamics. Although it may show higher MSE, in this case a smaller network might be a better choice, since the lower MSE of a larger network is a consequence of a better match to the noise fluctuations. It can be assumed that a smaller network, properly trained, shows an higher resilience to noisy training patterns. The second reason regards the generalisation ability of the network, i.e. its ability to estimate the correct output when new inputs, not included in the training set, are presented at its input. A large network may easily become too specialised and loose its ability to represent the general characteristics of the signal rather than the characteristics of the particular realisation given by the training set.

One possible strategy is to select an initially large set of nodes related to the training set (e.g. centres selected from the input set in a RBFN), and successively reduce this large number using a *pruning* algorithm. An alternative strategy is to start with a relatively small number of nodes, and successively add nodes according to some *growing* strategy. In this chapter, we will discuss a particular pruning technique, known as Orthogonal Least Squares algorithm, OLS (Chen *et al.*, 1989, 1991). This technique, applied to single-layer networks optimised in the MMSE sense, reduces the dimensionality of the $R^M \rightarrow R$ mapping in the linear layer and produces a corresponding reduction in the number of nodes.

4.2. Pruning algorithms

The input sequence of D-dimensional vectors $\mathbf{x}_n$ is mapped by the non-linear layer into the sequence of M-dimensional vectors $\mathbf{h}_n = [h_1(\mathbf{x}_n) \cdots h_M(\mathbf{x}_n)]^T$, for $n = 0 \ldots L-1$. The sequence $\mathbf{h}_n$ is linearly mapped into the output sequence $y_n = \mathbf{\Phi}^T \mathbf{h}_n$. It is realistic to assume that a large network always possesses a margin of redundancy in this mapping, and hence it is possible to find an optimal subset of M' nodes, $M' < M$, such that the performance using the mapping $R^{M'} \rightarrow R$ is relatively unaffected. In theory, the problem of finding a subset of nodes from an initially large population is difficult, since it requires the investigation of all possible combinations of M' nodes. A sub-optimal reduction may be obtained considering nodes individually, and at each step selecting the best node according to its contribution to the performance. In this way, the nodes are "ranked" according to their individual contributions with the lower rank nodes being removed to achieve the desired network size. This technique, referred as *forward selection*, it is sub-optimal in the sense that a group of two or more given nodes, whose combined contribution is large, might individually be ranked poorly with other nodes performing better than each single node in the group.

Such a ranking strategy is commonly referred to as *saliency of weights* (Bishop, 1995). After an initial training, the network is examined to assess the relative importance (saliency) of the weights. Typically some further training is required, and the procedure of training and pruning may be repeated for several cycles. There are various choices that can be made concerning the criterion used to assess the saliency. The simplest criterion is that smaller weights have smaller saliency, and therefore should be removed first. However, especially in non-linear networks, different nodes could have very different dynamic range, therefore the value of the weight might not reflect the saliency of the node.

A more principled approach is to consider the contribution of each node in the reduction of the cost function. Such an approach considers the change in the cost function due to small variations of the weights. The variation of the cost function C

due to the weight vector's variation $\mathbf{\Phi} + \delta\mathbf{\Phi}$, truncated at the second order, is given by:

$$\delta C = \sum_{i=1}^{M} \frac{\partial C}{\partial \phi_i} \delta\phi_i + \sum_{i=1}^{M} \sum_{j=1}^{M} \mathsf{H}_{ij} \delta\phi_i \delta\phi_j \tag{4.1}$$

Where H_{ij} are the elements of the Hessian matrix

$$\mathsf{H}_{ij} = \frac{\partial^2 C}{\partial\phi_i \partial\phi_j} \tag{4.2}$$

If the training process has converged, the first term in (4.1) vanishes. Hence the influence of each weight ϕ_i is determined by the i-th row of the Hessian matrix (4.2).

Le Cun *et al.* (1990) assume that the non-diagonal elements of the Hessian are null, and develop a pruning algorithm termed *optimal brain damage*. A technique that considers a non-diagonal Hessian matrix, termed the *optimal brain surgeon*, has been introduced by Hassibi and Stork (1993).

If the cost function is the MSE, the function $C(\mathbf{\Phi})$ is quadratic and (4.1) assumes a very simple form. In section 4.3 we will see that the contribution of the node (i.e. weight) to the minimisation of the MSE is given by its "self-contribution", corresponding to the diagonal elements of the Hessian matrix, and by its interaction with other nodes ("cross-contribution") corresponding to the non-diagonal elements. Both these factors should be taken into account if one is to consider the full effect of a node removal. In section 4.3 we will see how in OLS the orthogonalisation phase, based on the Gram-Schmidt algorithm, allows one to separate the two contributions, and plays a crucial role in the process of determining the real importance of each node.

4.3. MSE and orthogonality

The MMSE solution for the weight vector $\mathbf{\Phi}$ is given by (2.21). The problem addressed by a pruning algorithm is to decide which of the terms $h_i(\cdot)$ in (2.20) can be safely discarded with the minimum degradation of the MMSE. We denote the *i*-th column $\mathbf{p}_i = [h_i(\mathbf{x}_0)\cdots h_i(\mathbf{x}_{L-1})]^T$ of the intermediate space $\mathbf{H}$ the *i*-th *regressor* of the regression represented by equation (2.20). The regressor $\mathbf{p}_i$ is a vector in R^L. Given the desired output (target) sequence $\mathbf{T}$, it is easy to show that the contribution of the *i*-th regressor to the reduction of the MSE is proportional to a quantity called *Error Reduction Ratio* (Chen *et al*, 1989):

$$err_i = \left(\phi_i^2 \, \mathbf{p}_i^T \mathbf{p}_i + \sum_{j \neq i}^{M} \phi_j \phi_i \, \mathbf{p}_j^T \mathbf{p}_i \right) \left(\mathbf{T}^T \mathbf{T} \right)^{-1} \tag{4.3}$$

Note that the summation in (4.3) is a measure of the cross-correlation of the regressor *i* with all the other regressors. Hence, if we discard $\mathbf{p}_i$, we increase the MSE by the "self-contribution" (given by $\phi_i^2 \, \mathbf{p}_i^T \mathbf{p}_i$), but also by the "cross-contribution" $\phi_j \, \phi_i \, \mathbf{p}_j^T \mathbf{p}_i$. Hence, we also decrease the error reduction ratio of all the other regressors. Furthermore, the weight ϕ_i depends on the combined effect of all the regressors with the target sequence.

Clearly, the problem of determining the best regressor, at each step of the ranking procedure, is not separable unless the cross-correlation terms is removed. We can transform the intermediate space $\mathbf{H}$ into another space $\mathbf{W}$ with the same span as $\mathbf{H}$, where $\mathbf{W}$ describes a new set of *M* regressors $\mathbf{w}_i = [w_{i0} \ldots w_{i(L-1)}]^T$, $i = 1 \ldots M$ that satisfy an *orthogonality* condition, i.e. $\mathbf{w}_j^T \mathbf{w}_i = 0, \forall i \neq j$. In this case the cross-contributions in (4.3) vanish and the saliency of $\mathbf{w}_i$ does not depend on its interaction with the other regressors.

Thus it is necessary to devise a procedure that orthogonalises the regressors $\mathbf{p}_i$ in R^L. The well known Gram-Schmidt orthogonalisation algorithm, in its classical form (CGS) or in its modified form (MGS) (Chen *et al.*, 1989), provides an appropriate tool for this orthogonalisation. The MGS possesses superior numerical properties relative to CGS. The inclusion of a saliency criterion based on the error reduction ratio computed in terms of orthogonalised regressors forms the Orthogonal Least Squares Learning algorithm.

4.4. Orthogonal Least Squares Learning Algorithm

Orthogonal Least Squares Learning algorithm (Chen *et al*, 1989, 1991) is an effective technique to reduce the number of nodes in a linear-in-the-parameters network (like single-layer networks) by means of a Gram-Schmidt orthogonalisation, coupled with a saliency criterion that finds the optimal regressor at each step of the orthogonalisation procedure. This criterion used is to choose the orthogonal regressor at step k with the highest error reduction ratio (4.3), once the cross-contribution has been eliminated, and use this regressor as the k-th basis of the representation. We now describe the algorithm using the CGS algorithm:

1) At the first step, for each $i = 1 \ldots M$, we compute the *regressor's gain*:

$$g_i = \frac{\mathbf{w}_i^{(1)T}\mathbf{T}}{\mathbf{w}_i^{(1)T}\mathbf{w}_i^{(1)}} \tag{4.4}$$

where $\mathbf{w}_i^{(1)} = \mathbf{p}_i$. Then we compute the error reduction ratio:

$$err_i^{(1)} = \frac{g_i^2\,\mathbf{w}_i^{(1)T}\mathbf{w}_i^{(1)}}{\mathbf{T}^T\mathbf{T}} \tag{4.5}$$

and determine the first orthogonal basis $\mathbf{w}_{i_1}$ by choosing the index i_1 that maximises err_i

$$\mathbf{w}_{i_1} : \; i_1 = arg\left\{ \max_i (err_i) \right\}, i = 1 \ldots M \tag{4.6}$$

Let us consider equations (4.5) and (4.6) in more detail. Each regressor $\mathbf{p}_i$ is considered as if it was the only element in the regression series. In other words, at the first step the algorithm selects the best single regressor that approximates the desired target signal **T**. The regressor gain (4.4) the MMSE solution (weight) for a network made of a single node.

2) At the k-th step, where $k \geq 2$, for $i = 1 \ldots M$, $i \notin \{ i_1 \ldots i \neq i_{k-1} \}$, $i_1 \leq j \leq i_{k-1}$, compute:

$$\alpha_{ji}^{(k)} = \frac{\mathbf{w}_j^T \mathbf{p}_i}{\mathbf{w}_j^T \mathbf{w}_j} \tag{4.7a}$$

$$\mathbf{w}_i^{(k)} = \mathbf{p}_i - \sum_j \alpha_{ji}^{(k)} \mathbf{w}_j \tag{4.7b}$$

These equations have a simple interpretation. Equation (4.7b) orthogonalises each remaining candidate $\mathbf{p}_i \quad i = 1 \ldots M$, $i \neq i_1 \ldots i \neq i_{k-1}$ with respect to the set of orthogonal regressors already found, $\mathbf{w}_{i_1} \ldots \mathbf{w}_{i_{k-1}}$. To obtain this orthogonal vector, the projection of $\mathbf{p}_i$ over $\mathbf{w}_{i_1} \ldots \mathbf{w}_{i_{k-1}}$, is equal to $\sum_j \alpha_{ji}^{(k)} \mathbf{w}_j$, and is subtracted from the non-orthogonal regressor $\mathbf{p}_i$. Equation (4.7a) is an ancillary equation that calculates the projection of $\mathbf{p}_i$ over $\mathbf{w}_j$ (scalar product, divided by the squared norm of $\mathbf{w}_j$).

The k-th orthogonal regressor will be chosen using the following equation:

$$err_i^{(k)} = \left(g_i^{(k)}\right)^2 \cdot \frac{\mathbf{w}_i^{(k)T}\mathbf{w}_i^{(k)}}{\mathbf{T}^T\mathbf{T}} \tag{4.8a}$$

where

$$g_i^{(k)} = \frac{\mathbf{w}_i^{(k)T}\mathbf{T}}{\mathbf{w}_i^{(k)T}\mathbf{w}_i^{(k)}} \tag{4.8b}$$

And by substituting (4.8b) in (4.8a) we can express the error reduction ratio in terms of the *coherence function*:

$$err_i^{(k)} = \frac{\left(\mathbf{T}^T\mathbf{w}_i^{(k)}\right)^2}{\left(\mathbf{w}_i^{(k)T}\mathbf{w}_i^{(k)}\right)\left(\mathbf{T}^T\mathbf{T}\right)} \tag{4.8c}$$

We can either stop the algorithm when we get the required number of regressors (i.e. nodes) or alternatively when we get a satisfactory MSE (compared with the MSE obtained using the original full set of regressors).

It can be easily shown that the orthogonalised problem corresponds to a MSE equation where the cross-correlation matrix $\mathbf{W}^T\mathbf{W}$ is diagonal. Furthermore, the columns of $\mathbf{H}$ (the original regressor set) will be related to the columns of $\mathbf{W}$ by an upper triangular matrix:

$$\mathbf{H} = \mathbf{W}\cdot\mathbf{A} = \mathbf{W}\cdot\begin{bmatrix} 1 & \alpha_{12} & \dots & \alpha_{1M} \\ 0 & 1 & \dots & \alpha_{2M} \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} \tag{4.9}$$

Equation (4.9) is the key to obtain the desired reduction, since the reduced orthogonal set $\mathbf{W}'$ with $M' < M$ corresponds to an equally reduced, non-orthogonal set $\mathbf{H}'$.

A relation similar to (4.9) may be obtained, that relates the MMSE weights ϕ_i of the non-orthogonal set to the weights g_i, through the matrix **A**. Alternatively, one may choose to recalculate the weights of the reduced network using equation (2.21).

The previous formulation of the OLS algorithm is based on the classical form of the Gram-Schmidt algorithm. In this formulation, at each step k the remaining regressors are orthogonalised with respect to the previously found k-1 orthogonal regressors, and then the k-th orthogonal regressor is found. Experimental results (Rice, 1966) and the theoretical analysis (Björck, 1967) show that, if **H** is ill-conditioned, the columns of **W** will soon loose their orthogonality and re-orthogonalisation becomes necessary.

The modified Gram-Schmidt (MGS) orthogonalisation (Björck, 1967) possesses superior numerical qualities (Chen *et al*, 1989). Once the (k-1)-th regressor have been found, the remaining columns of **H** are orthogonalised, and the k-th regressor is found. The first step of the OLS based on MGS is the same as that used in the CGS procedure, according to equations (4.4), (4.5) and (4.6). However, using MGS the remaining M-1 non-orthogonal regressors are orthogonalised with respect to $\mathbf{w}_{i_1}$ prior to the next iteration. Hence at the k-th step, the regressors have been already orthogonalised to the previous k-2 bases. Hence equation (4.7b) is reduced to:

$$\mathbf{w}_i^{(k)} = \tilde{\mathbf{p}}_i - \alpha_{i_{k-1}i}^{(k)} \mathbf{w}_{i_{k-1}} \tag{4.10a}$$

where $\tilde{\mathbf{p}}_i$ denotes the regressor that has been orthogonalised with respect to the previous k-2 orthogonal bases. The target sequence is also orthogonalised:

$$\tilde{\mathbf{T}} = \mathbf{T} - g_{i_{k-1}} \mathbf{w}_{i_{k-1}} \tag{4.10b}$$

It is important to note that, in absence of round-off errors, the two procedures are equivalent.

4.5. Matrix form of the OLS algorithm

It is convenient to express the OLS algorithm, in its MGS form, directly in terms of the matrices **R** and **P** as in (2.21). Specifically in terms of their elements r_{ij} and p_i. Equation (4.4) can be expressed as:

$$g_i = p_i / r_{ii} \tag{4.11}$$

and consequently the error reduction ratio can be computed as follows:

$$err_i = p_i^2 / r_{ii} \tag{4.12}$$

Note how these two quantities depend on the diagonal elements of **R**. The MGS orthogonalisation is calculated on the i-th column and row of **R** and the i-th element of **P**. At the k-th step the projection of the i-th candidate (4.7a) is calculated as:

$$\alpha_{ji}^{(k)} = r_{ji} / r_{jj}, \quad j = i_{k-1}, \quad i \neq i_1 \ldots i_{k-1} \tag{4.13}$$

The elements $r_{il}, p_i, i, l \neq i_1 \ldots i_{k-1}$ of **R** and **P** are orthogonalised using:

$$\begin{aligned} r_{il}^{(k)} &= r_{il}^{(k-1)} - \alpha_{ii_{k-1}} \alpha_{li_{k-1}} r_{i_{k-1}i_{k-1}} \\ p_i^{(k)} &= p_i^{(k-1)} - \alpha_{ii_{k-1}} p_{i_{k-1}i_{k-1}} \end{aligned} \tag{4.14}$$

The matrix formulation of the OLS algorithm is useful when the direct computation of the correlation matrices **R** and **P** is impractical. For instance, if the regressor's length L is excessive the computation the scalar product $r_{ij} = \mathbf{w}_i^T \mathbf{w}_j$ for the whole sequence imposes a heavy computational load. In this case, the computation of the correlation matrices can be split into smaller training sets, and the overall result is obtained by summing the results and applying OLS directly on the final matrices.

4.6. Conclusions

The computational effort is a factor of paramount importance when designing a de-interlacing system. Single-layer networks easily incur in the so-called "curse of dimensionality" by delivering performance improvement at the cost of increased complexity. Improper training may lead to excessively large networks. A possible strategy is to train an initially large network and successively to remove those nodes whose contribution to the goal is somehow least relevant. Such a strategy is known as saliency of weights. If the cost function is the MSE, it is easy to determine a saliency function for the linear parameters. The Error Reduction Ratio in (4.3) accounts for the individual contribution of each node to the reduction of the MSE.

In order to select an optimal subset, one should consider all the possible combinations of nodes. This is reflected in (4.3), since the contribution of the individual node depends on the other nodes. A sub-optimal strategy is represented by Orthogonal Least Squares techniques. A Gram-Schmidt based orthogonalisation is coupled with the Error Reduction Ratio in order to produce a triangular decomposition of the normal equation where the orthogonal basis have decreasing saliency according to (4.3). The principle is that at each step of the algorithm, each individual node is assessed in its ability to reduce the MSE.

Some care must be taken in order to ensure a numerically robust algorithm. This is because as the orthogonalisation proceeds, the remaining regressors yield output values closer to the limits of the available numerical range. It has been observed that the Modified Gram-Schmidt algorithm is less prone to this kind of problems. A matrix formulation of the algorithm is also provided that allows the orthogonalisation of long training sequences of data to be split in smaller batches of data.

5. WIENER AND VOLTERRA DE-INTERLACING

5.1. Introduction

In this chapter we will discuss the results obtained by de-interlacing the picture in figure 1 using Wiener linear filters and Volterra series. The odd field and the even field are obtained from the odd and even rows of the picture. The even field is sampled to produce the input set, and the even field is used to obtain the corresponding target set.

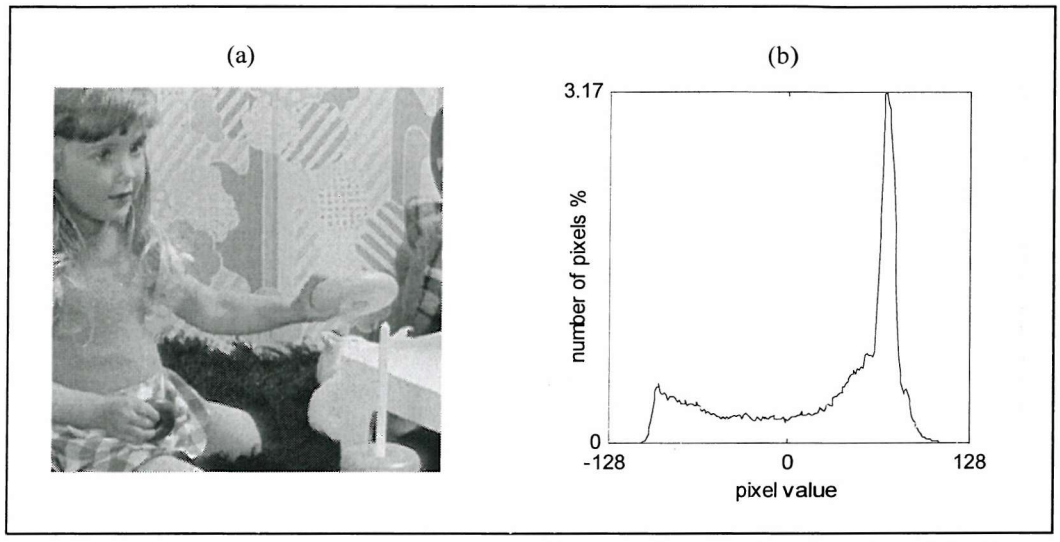

Figure 1. (a) "Girl" picture (301x301 pixels). (b) histogram (percent of total number of pixels).

In choosing an appropriate de-interlacing system based on these two techniques, one is presented with the choice of different architectural alternatives. The choice of a proper filter aperture D is an immediate concern in the design of a de-interlacing system. Realistic systems are limited by computational constraints, usually determined by speed requirements and hardware limitations. Therefore, it is of no use to obtain good performance with a broad aperture, if this choice leads to an unfeasible architecture. Moreover, it will be shown that, at least for the linear case, there is an aperture limit beyond which there is no significant gain in performance. The Wiener and Volterra interpolators are tested with the apertures depicted in figure 2. Note that in some cases, the same number of pixels is used in different apertures. The aim of this is to show the relative importance of vertical and horizontal information in the recovery of the original field. It will be shown that the vertical axis yields most of this

relevant information for the linear case. However in the application of the Volterra series, the horizontal taps will play an important, albeit secondary, role. Another point of concern with Volterra series is the choice of the order p. In chapter 2 we have seen how p determines, together with the sampling aperture D, the number of branches in the Volterra network.

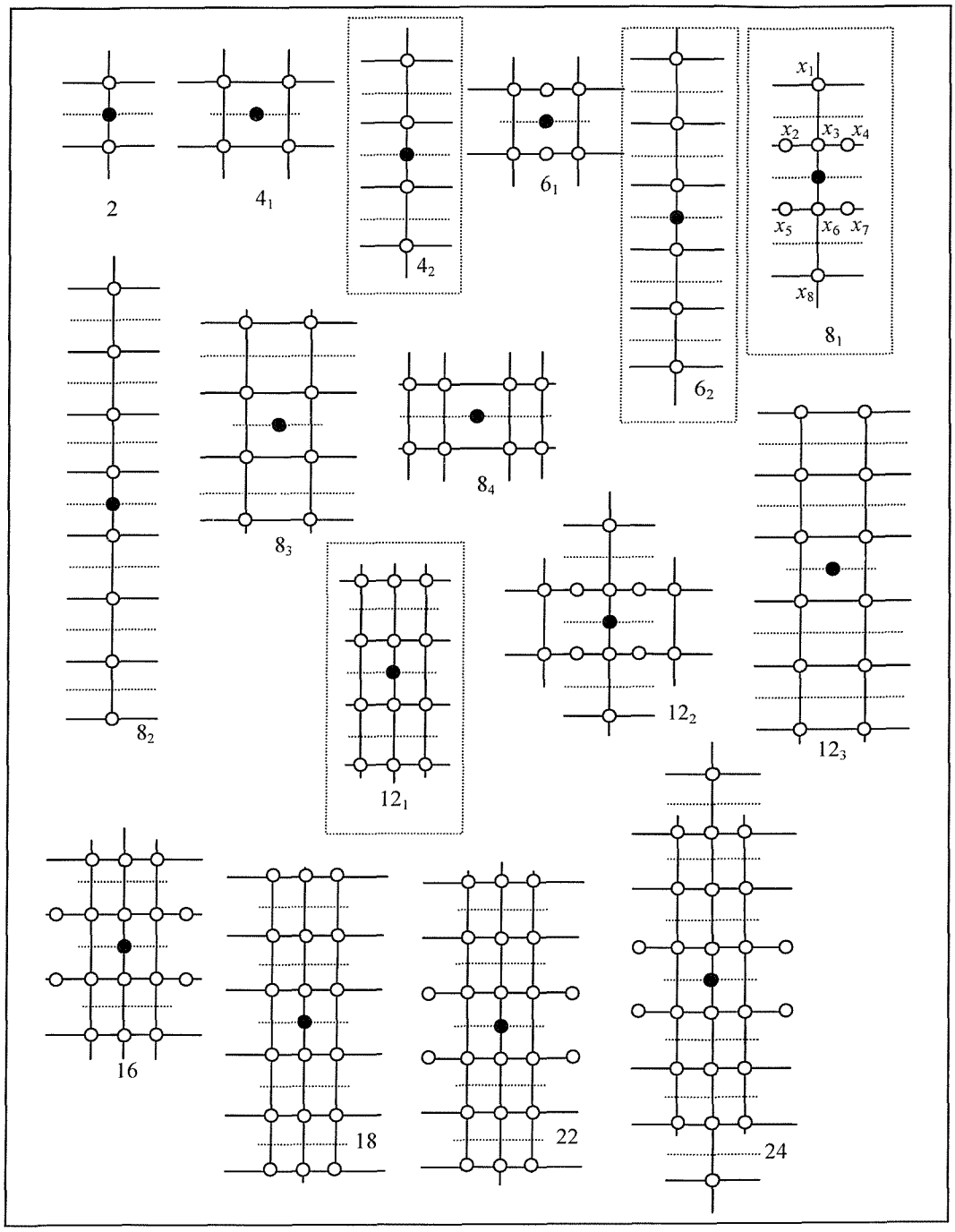

Figure 2. Sampling apertures. The black pixel is the pixel to be estimated. There are various apertures using the same number of pixels. In each case, the best shape (see table 1) is highlighted using a dotted frame. The pixel numbering of the elements of the input vector **x** is from left to right, top to bottom as shown for aperture 8_1.

The relationship between the order of the series and the order of the statistics of the training signal has also been discussed. This relationship helps us to determine a reasonable order for a Volterra series that produces a feasible de-interlacing system. In figure 1 the histogram of the single pixels comprising the picture, normalised to the number of pixels, is plotted. Clearly the pixel histogram can not produce an estimate of the joint input-target density, $p(\mathbf{x},t)$ which is the problem we seek to solve. However, it does produce an estimate of all the marginal distributions of $p(\mathbf{x},t)$. We can clearly see how the histogram is skewed and plati-kurtic (table 1). Hence it is reasonable to assume that the unknown distribution ought to be described at least in terms of its fourth-order moments, which implies that at least a third-order Volterra series should be implemented.

	Gaussian N(0.1299, 0.1522)	"Girl" (normalised)
mean	0.1299	0.1299
variance	0.1522	0.1522
skewness	0	-0.2827
kurtosis	3	2.4549

Table 1. Higher-order moments for the picture "girl" (normalised) compared to the moments of a Gaussian distribution with the same mean and variance.

5.2. Wiener Linear Filters

The results for the Wiener filters (2.12) with the various apertures in figure 2 are shown in Table 2 and the MSE plot is shown in figure 3 (where for a given aperture size only the result from the best shape is plotted). Regardless of the aperture's shape and size, it is evident that most of the information is drawn from the closest vertical neighbours of the target pixel, see figures 4 and 5. In fact, a reasonable linear interpolator merely computes an average of these two values. Note the small MSE improvement when apertures with some horizontal extent are used (e.g. 8_1) compared with purely vertical apertures (e.g. 8_2). This implies that horizontal information plays a minor role in the estimation of the output. On the other hand, apertures that do not use vertical information (e.g. 8_3 and 8_4) produce worse results.

	2	4_1	4_2	6_1	6_2	8_1
bias	-0.2100	-0.3057	0.0345	-0.1619	-0.0211	0.0266
MSE	1.2369	2.7737	1.0083	1.1916	0.9971	0.9893

	8_2	8_3	8_4	12_1	12_2	12_3
bias	-0.0040	-0.0661	-0.0469	0.0282	0.0251	-0.0880
MSE	0.9962	2.5169	2.3406	0.9784	0.9885	2.5147

	16	18	22	24
bias	0.0277	-0.0193	-0.0198	-0.0041
MSE	0.9773	0.9655	0.9634	0.9624

Table 2. Output MSE, Wiener filter (bold: optimal shapes).

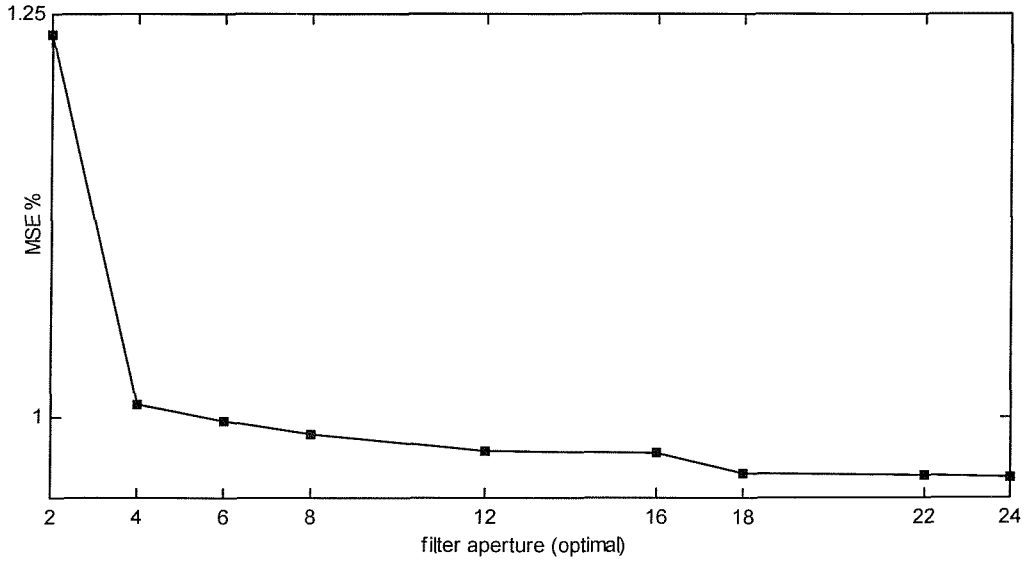

Figure 3. Output MSE vs. filter's aperture.

The smoothing artefacts introduced by linear filters can be readily explained. The effect of a linear filter is that the outcome will be roughly equal to the averaged luminosity of the closest neighbours. If these differ by a significant amount, like for example in a high-contrast area of the image, the resulting output will have an intermediate value, producing the smoothing effect. The "jagging" artefacts are a peculiarity of de-interlacing on oblique edges. Since only the interpolated field will be smoothed, that will create a pattern of alternating sharp and smoothed edges.

One can see an overall 22% MSE improvement in moving from a 2 to 24 pixel aperture, with an associated increase of 1100% in the number of pixel values processed. Moreover, the performance (figure 3) rapidly flattens and there appears to be no reason to assume that a further increase in the aperture size will yield significant increases in performance.

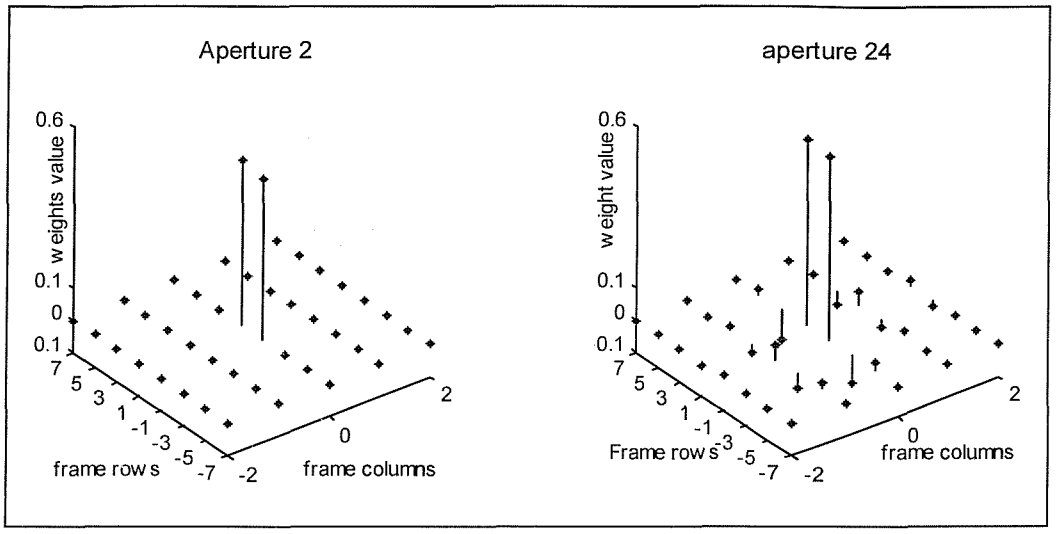

Figure 4. Filter weights for 2 and 24 pixel apertures.

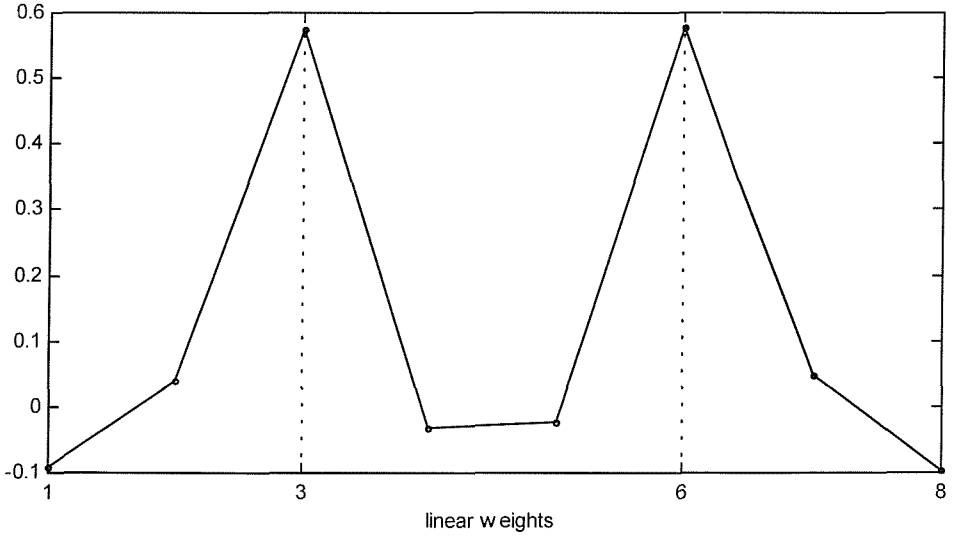

Figure 5. Wiener optimal weights for aperture 8_1, plotted on a one-dimensional axis.

It is also evident that the influence of the bias is negligible. In 8 bit precision, we can assume that the bias is zero for any aperture. This is easy to explain, considering that the bias accounts for the difference between the average value of the vertical

neighbours and the average of the target. Since these pixels come from the same frame, one might anticipate that this difference is close to zero.

5.3 Volterra interpolation

The results for a 3rd order Volterra deinterlacer employing each of the sampling schemes proposed in figure 2 are shown in table 3. Although powerful, Volterra series are computationally demanding and so the number of pixels in the aperture and the degree of the polynomial must be limited. It has been assumed that a reasonable model is obtained by a third order series. Collis *et al.* (1997) suggests that an aperture like 8_1 and a 3rd degree Volterra series (164 nodes) is likely to be an affordable target in terms of hardware. A 12-taps aperture requires 454 nodes giving a relatively modest increase of the performance (table 3). An 8-pixel, fifth order series will have the unrealistic number of 1286 nodes. For these reasons, we will consider orders no higher than 3. Furthermore, the aperture 8_1 will be the only aperture considered in the following chapters.

The results for the Volterra series are compared with those for linear interpolation in figure 6. Some qualitative comparison between the two techniques is given in figures 7.a and 7.b. It is evident that the Volterra deinterlacer performs significantly better than the linear filter. However note how the rate of improvement over the linear filter decreases when the aperture becomes larger than 4. In the larger apertures many coefficients prove to be redundant (Section 5.4). Table 3 indicates that apertures with horizontal taps alongside the vertical ones are more effective than in the linear case, for example comparing the performance of apertures 8_1 and 8_2 with the corresponding performance with linear interpolation. Horizontal information plays a more important role in the higher order kernels.

Figure 8 shows the absolute value of the linear coefficients of the Volterra model for the aperture 8_1. These appear very similar to the isolated linear filter (figure 5). The full Volterra coefficients for apertures 2, 4_2, 8_1 and 18 are plotted in figure 9a and 9b. When examining figure 9 one should realise that the coefficients are on different scales depending on the kernel order they refer to.

		lattice					
		2	**4_1**	**4_2**	**6_1**	**6_2**	**8_1**
Number of coefficients/ degree	**1st**	2	4	-	6	-	8
	2nd	3	10	-	21	-	36
	3rd	4	20	-	56	-	120
MSE		1.1082	1.1082	0.7850	0.9451	0.7575	0.6956

	8_2	**8_3**	**8_4**	**12_1**	**12_2**	**12_3**
1st	-	-	-	12	-	-
2nd	-	-	-	78	-	-
3rd	-	-	-	364	-	-
MSE	0.7432	2.1210	1.8335	0.6521	0.6715	2.0314

	16	**18**	**22**	**24**
1st	16	18	22	24
2nd	136	171	253	300
3rd	816	1140	2024	2600
MSE	0.5947	0.5751	0.5025	0.4676

Table 3. Output MSE, 3rd order Volterra series. The best results for each aperture are highlighted.

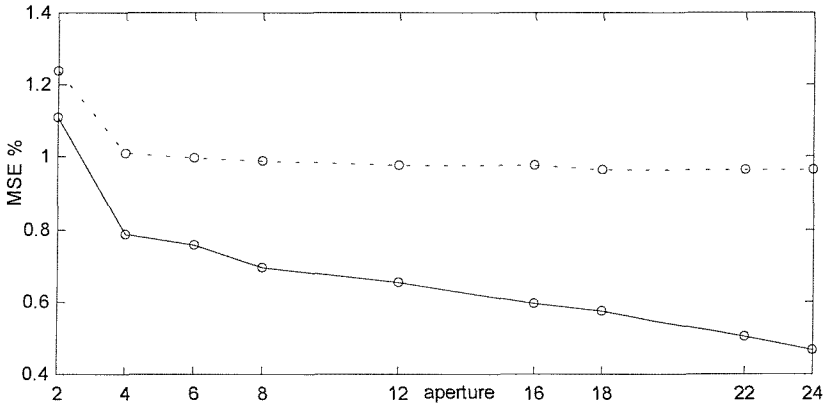

Figure 6. Volterra (—) and Wiener (--) de-interlacing.

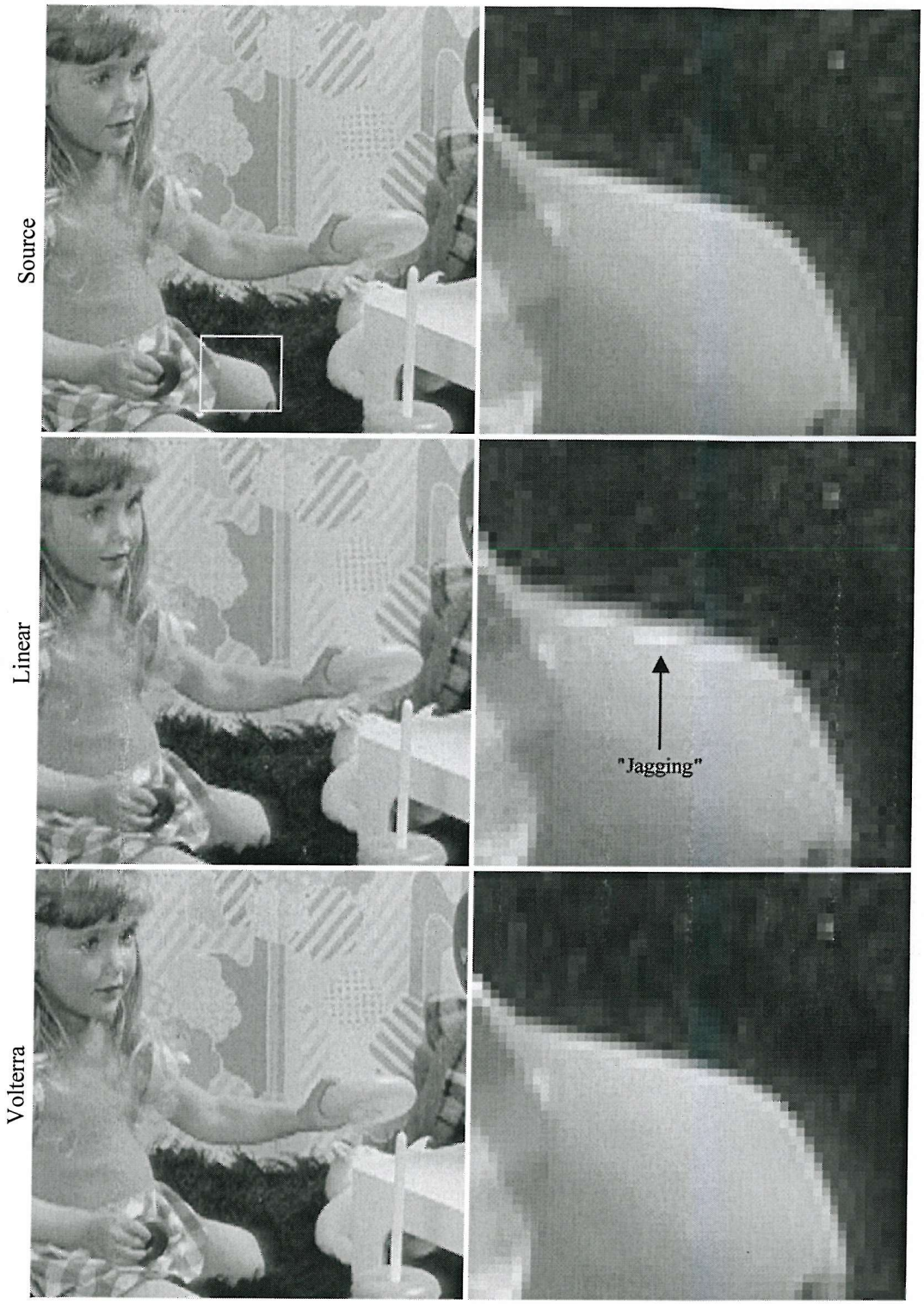

Figure 7.a. "jagging" artefacts in linear interpolation and edge preserving properties of Volterra interpolation. Aperture 8_1 has been used for both systems.

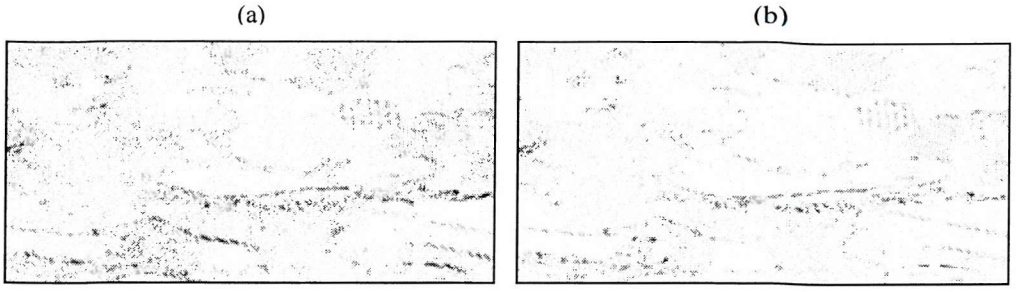

Figure 7.b. Difference field with respects to the target field. (a) Linear filter (b) Volterra series

The reason for this characteristic outcome is partially to be found in the dynamics involved in the solution and application of the Volterra series. It is hence incorrect to deduce, from the analysis of the weights alone, that the linear kernel is more important than the quadratic and the cubic kernels. In the linear case, the regressor **h** spans a cube in R^D bounded by $[-127,128]$. In the Volterra case, second order regressors will span a subspace bounded by $[-127^2,128^2]$ and third order regressors will be bounded by $[-127^3,128^3]$.

In fact, if we take into account the different dynamics of the first, second and third order nodes we have a different picture about the relative importance of the kernels. In figure 10 the dynamics of the matrices **P** and **R** are shown, and it is possible to appreciate the effect of varying scales in the matrix. Figure 11a shows the different dynamics of the matrix **R** calculated for the aperture 8_1. The brackets indicate the kernel order. It is evident how the 3rd-order dynamics dominate the dynamics of the first and second order kernels. Figure 11b, where **R** is normalised to the local dynamic range, shows a clearer picture of the relative magnitude of each order of kernels. However, it would be incorrect to draw any assessment about the relative importance of each kernel from this picture. One should remember that figure 10b is normalised to the local dynamics. Furthermore, we have not performed a similar analysis on the cross-correlation vector **P**. A more structured approach is presented in the next section, where OLS reduction is applied to the series.

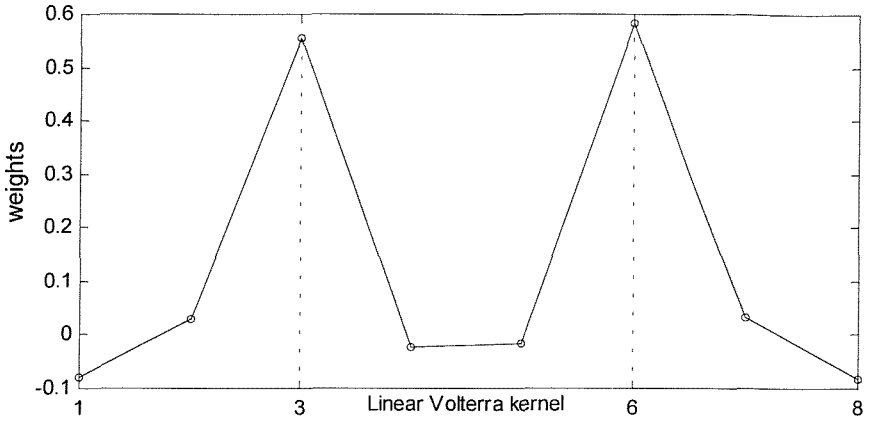

Figure 8. Volterra linear coefficients (aperture 8_1), plotted on a one-dimensional axis.

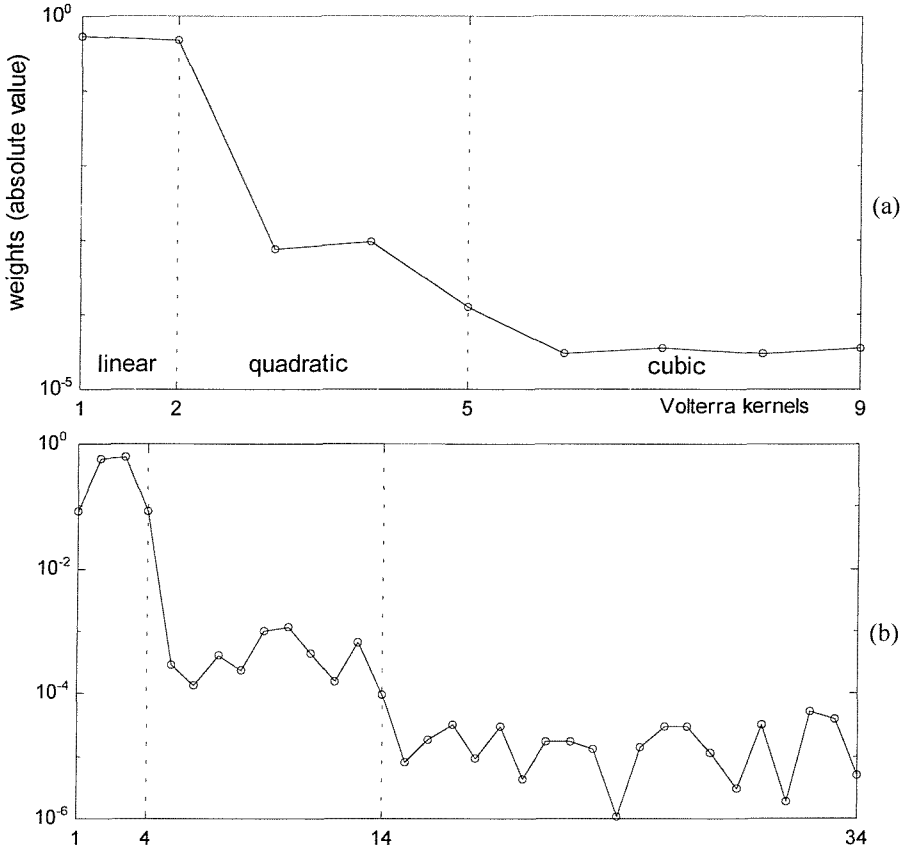

Figure 9a. Volterra kernels for apertures 2 (a) and 4_2 (b).

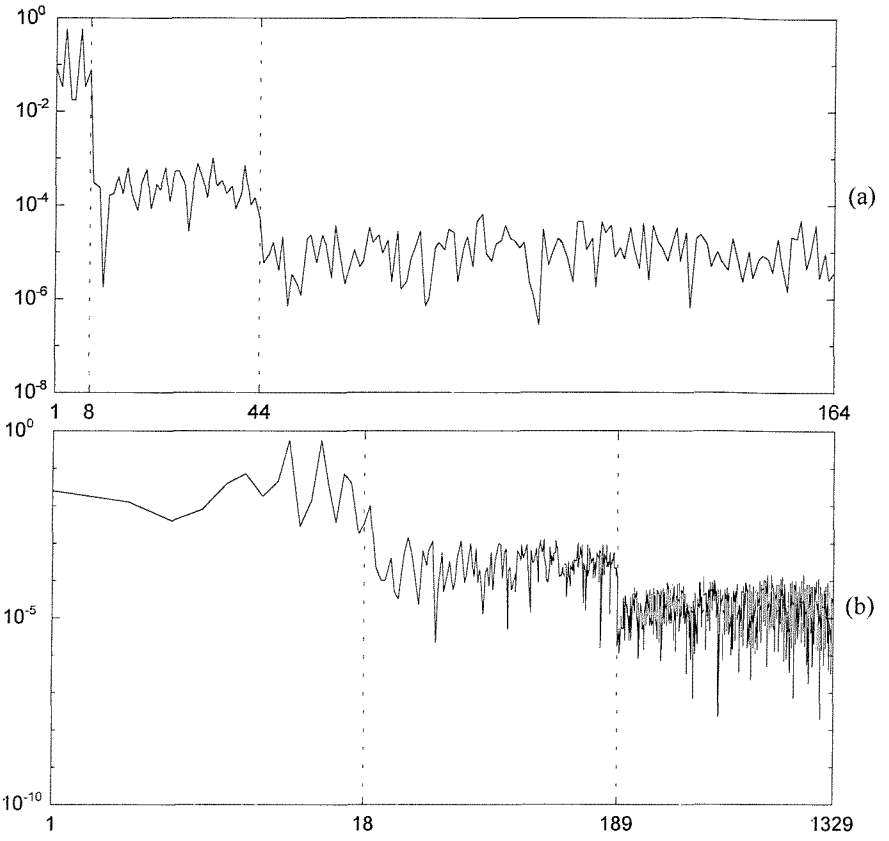

Figure 9b. Volterra kernels for apertures 8_1 (a) and 18 (b) (log x-axis).

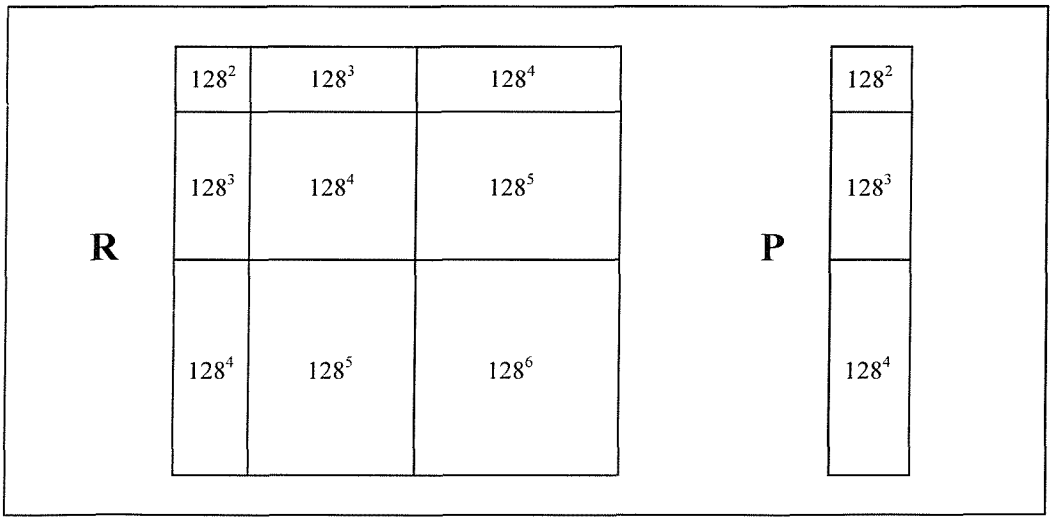

Figure 10. Dynamics of **P** and **R** for 3rd order Volterra series.

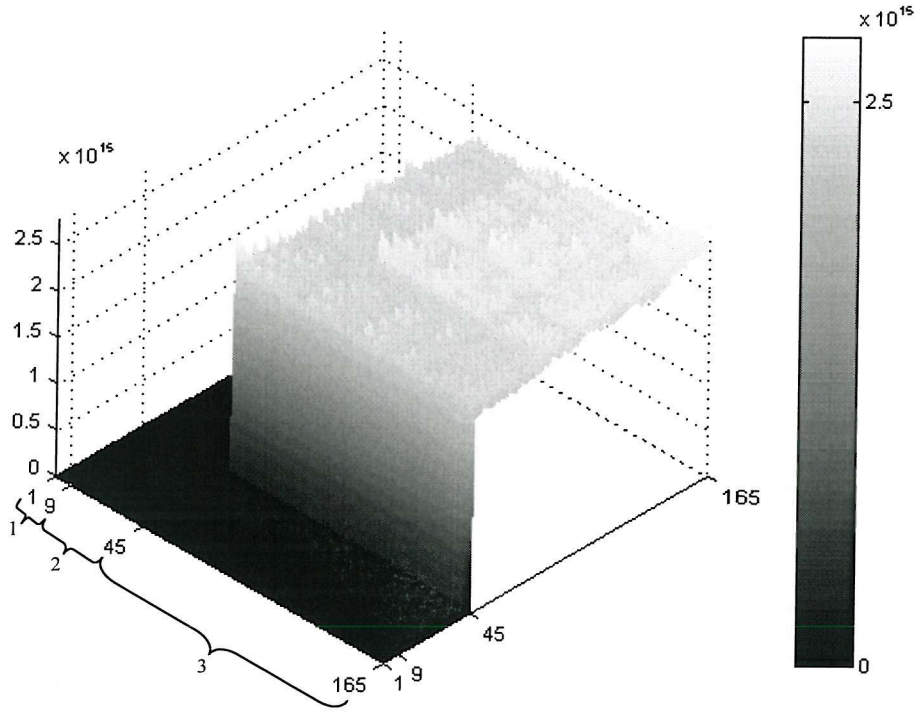

Figure 11a. Cross-correlation matrix **R** of the Volterra kernels, aperture 8_1.

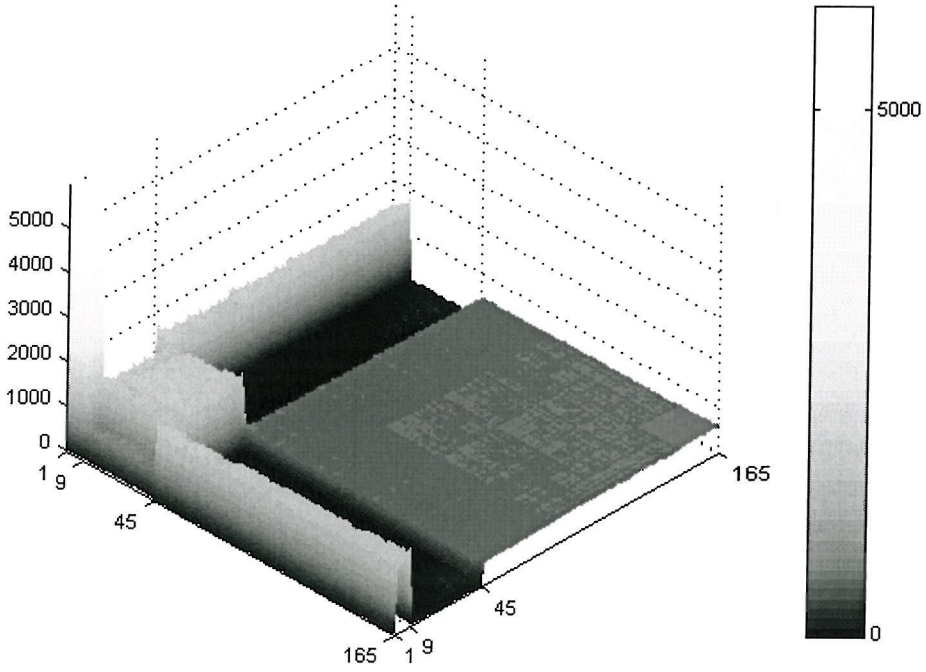

Figure 11b. Same matrix as in figure 10a, normalised to the relative dynamics.

5.4. Orthogonal Least Squares Volterra series

It is reasonable to question whether the Volterra model gives a correctly sized network, i.e. if the system, whose number of nodes is determined a priori by the degree and aperture size, rather than by the observed complexity of the data, is redundant. From a visual analysis in figure 11b, clearly appears that the regressors are strongly correlated. It is natural to question whether OLS might lead to solutions using a reduced number of nodes but yielding the similar MSE as the full Volterra scheme.

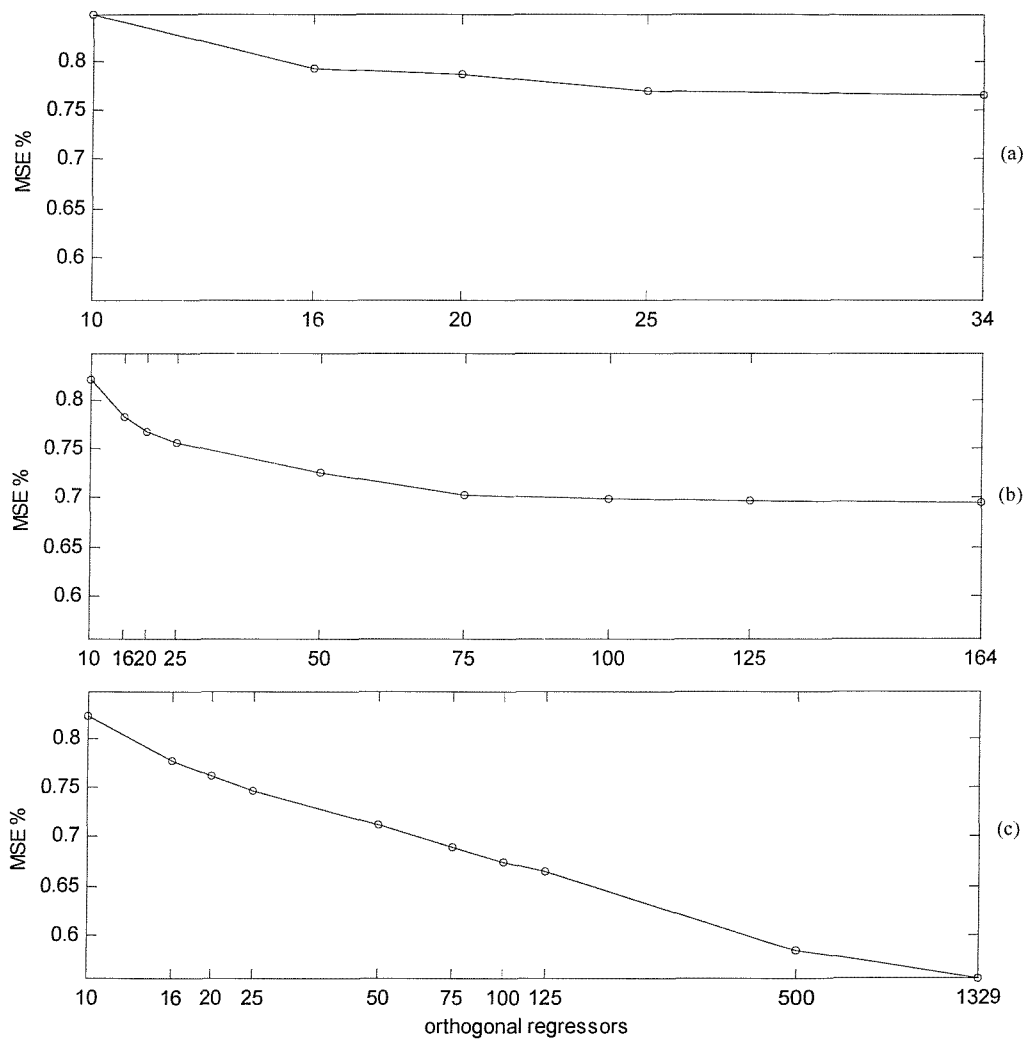

Figure 12. Orthogonal MSE sequence for the Volterra series, apertures 4_2 (a), 8_1 (b) and 18 (c).

In figure 12 we show the MSE for the OLS Volterra series with apertures 4_2, 8_1 and 18. It is apparent that the corresponding series are highly redundant, and an advantage in terms of efficiency of the network may be obtained by using OLS. However, the picture shown in these plots may be misleading when we want to estimate the corresponding reduction in the computational load required. In figure 13, the degrees of the orthogonalised regressors are shown. As one can see, although the first four regressors are always linear, the successive regressors are mainly third-order regressors. Hence, the reduction in the network's size does not correspond to an equal reduction in the computational load, since although we can use a smaller number of nodes to obtain approximately the same MSE, these nodes will be largely drawn from the third-order kernel, that requires the highest computational effort. This point will be further discussed in chapter 6, where we will compare the performance of RBFN and Volterra series.

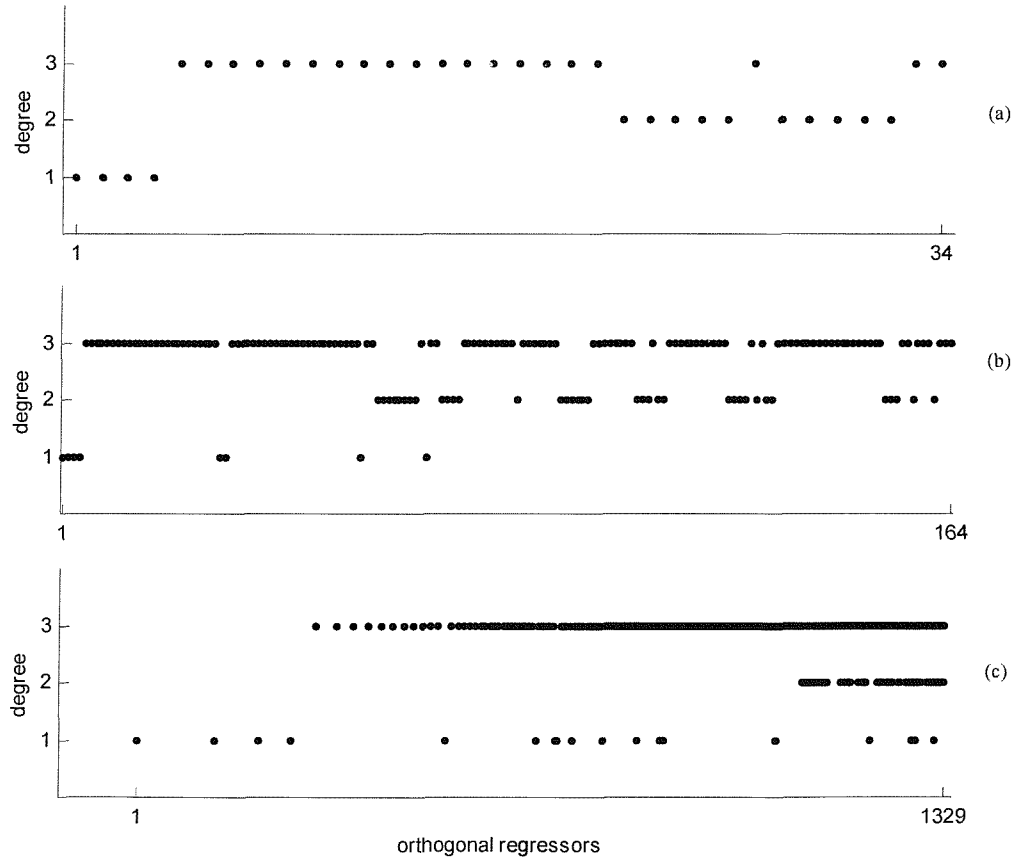

Figure 13. Degree of the orthogonal Volterra regressors, apertures 4_2 (a), 8_1 (b) and 18 (c).

Figure 13 confirms our early observation that the de-interlacing problem should be addressed by a series whose order is at least 3. Moreover, they show how the second-order kernels play a lesser role, compared to linear and cubic ones, in minimising the MSE. In figures 14a, 14b and 14c we can see the composition (degree and lattice position) of the monomials forming the first ten orthogonal regressors, for the three apertures considered.

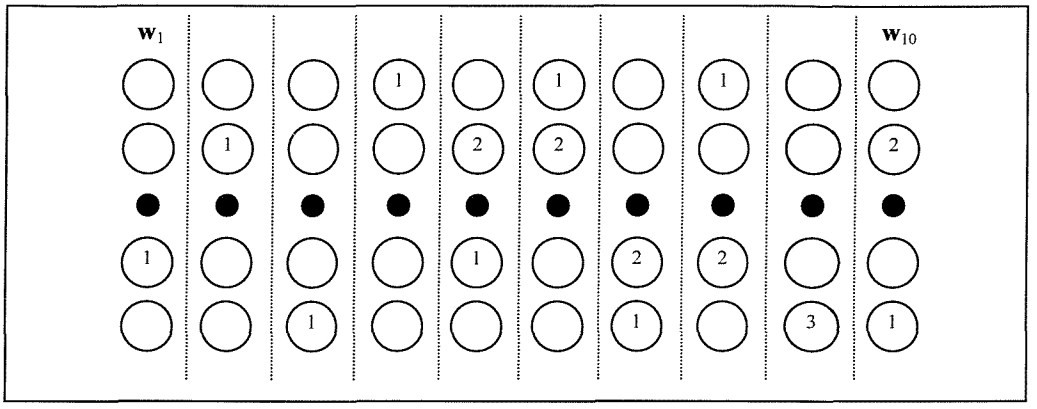

Figure 14a. Monomial composition of the first 10 orthogonal regressors, aperture 4_2. The number in the circles show which pixels are used in the regressor, with their degree in the monomial.

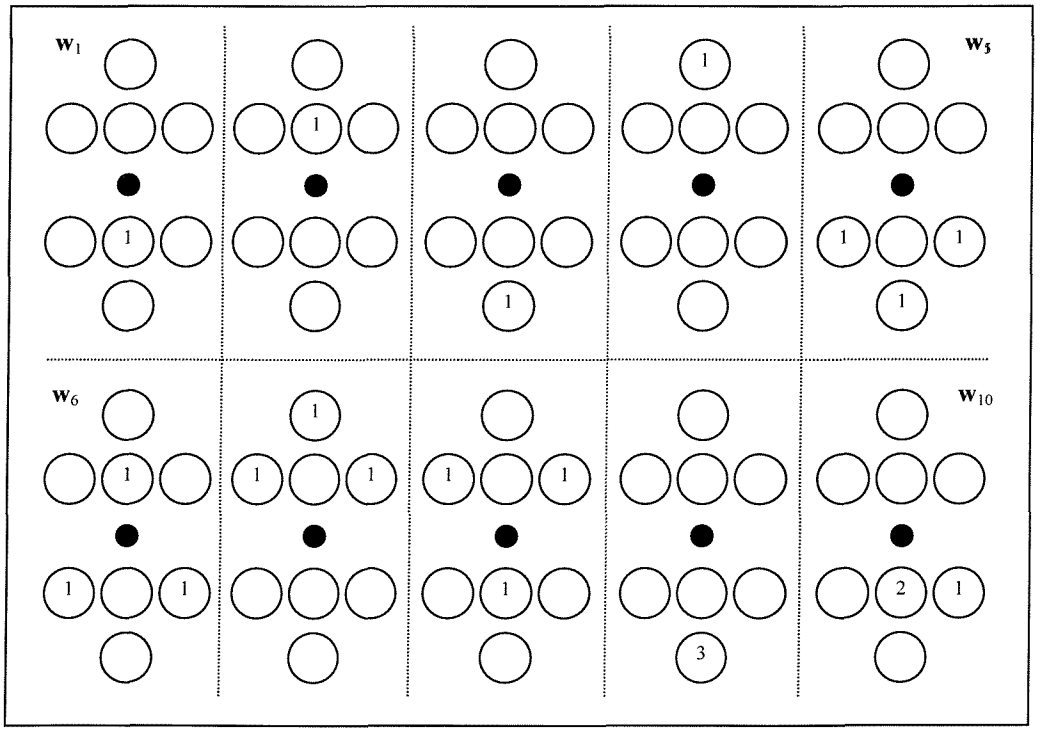

Figure 14b. Monomial composition of the first 10 orthogonal regressors, aperture 8_1.

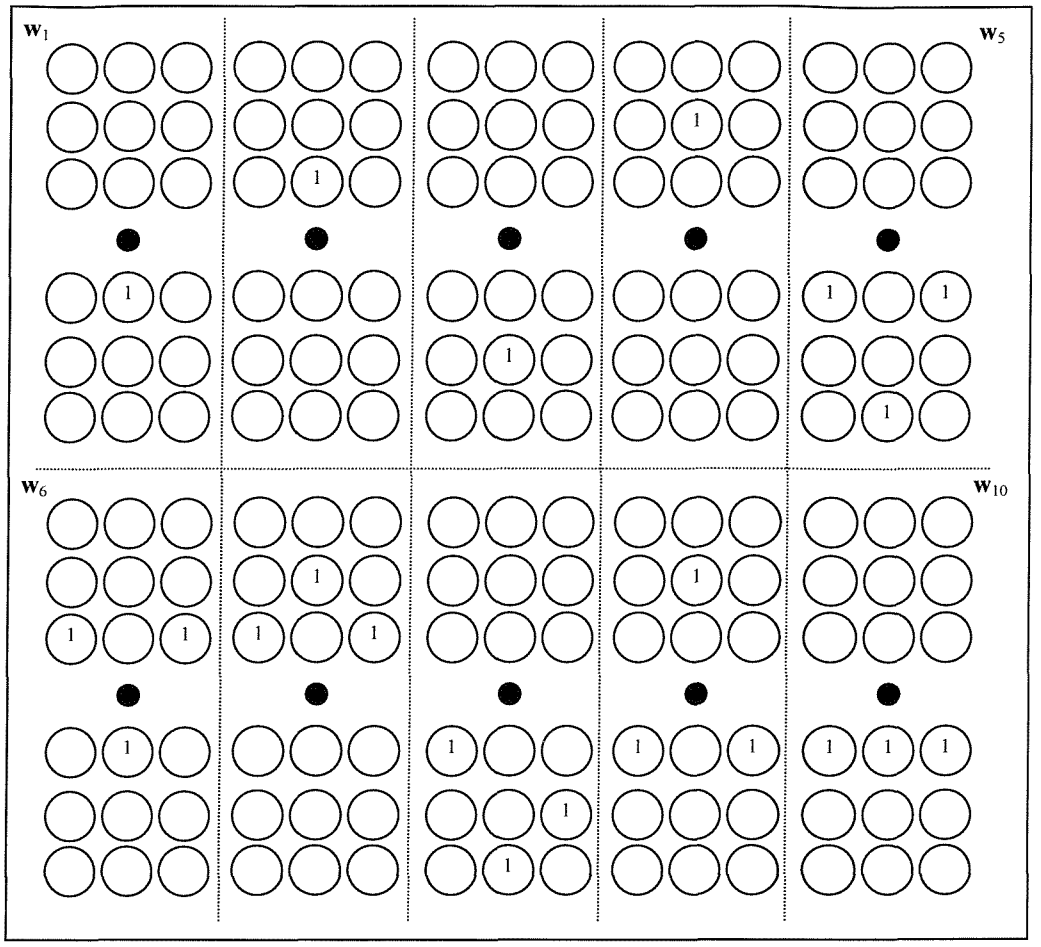

Figure 14c. Monomial composition of the first 10 orthogonal regressors, aperture 18.

In each figure the values in the surrounding pixels indicate the pixel degree in the monomial. It is confirmed that the four closest vertical neighbours play the most important role. Note also how, in the composition of the third-order nodes, some vertical information, coming from the four closest vertical taps, is always present.

However, in the case of the apertures 8_1 and 18, this vertical information is "modulated" by some horizontal information. This observation confirms the importance of horizontal information in reducing the MSE. We could view these third-order nodes as being first-order, with adaptive weights formed by the product of the MMSE weight and some other taps, mainly horizontal. It appears that the vertical information is adapted to some local measure of contrast/luminosity. A

comprehensive analysis of the nodes' composition in terms of their importance is however beyond the scope of this thesis.

5.5. Conclusions

This chapter has provided a significant overview on the application of linear and polynomial models to spatial de-interlacing. The experiments clearly show how Volterra-based de-interlacing produces a superior result compared to linear filters. They also provide some interesting insights into the way Volterra series exploit their internal structure to produce the mapping.

Firstly, a set of 16 sampling lattices has been selected to carry the experiments. The apertures increase from 2 to 24 pixels, and for each aperture several pixel arrangements are presented by stretching the lattice either in the vertical or the horizontal direction. The goal is to show how vertical and horizontal information play different roles in reducing the error.

These 16 considered in the experiments sampling lattices generate the training sets used to calculate a corresponding number of linear filters. The results clearly show the main limitations of linear interpolation. It is clear from the MSE plot that the error curve quickly flattens and there is little gain in further increasing the aperture. This is dramatically reflected in the much higher value of the weights corresponding to the vertical pixels, most noticeably the two close vertical neighbours. It is clear that the output value is mainly dependent on local vertical information and that explains the rapid flattening of the MSE curve. The resulting linear filters basically compute a local vertical average. The immediate effect is that contrasted areas get blurred. The alternation of sharp (source) and blurred (interpolated) lines in the de-interlaced frame eventually produces the observed "jagging" artefacts.

The picture changes when Volterra series are applied. The blurring is clearly reduced and the jagging mitigated. This time the experiments show that the horizontal information play a major role in achieving a much improved performance. Although the linear kernel remains basically unchanged, the plot of the higher order kernels shows a rich set of inter-pixel dependencies.

The application of OLS shows that the Volterra series is redundant. This is due to the way the series is initially built by fixing the degree p and the aperture D. OLS shows how the cubic terms play a major role in the error reduction. The observation of the first orthogonal regressors suggests an interpretation of the Volterra series in terms of adaptive filters, where the horizontal pixels provide local measurements of contrast and luminosity to change the values of a vertical linear filter.

6. GAUSSIAN RBFN DE-INTERLACING

6.1. Introduction

In the previous chapter we have discussed the application of linear filters and Volterra series to de-interlace a single frame. In this chapter we will discuss the application of Gaussian Radial Basis Function networks (GRBFN) to de-interlace that same frame.

RBFN differ in many aspects from polynomial techniques like the Volterra series, as already discussed in chapter 2. In Volterra series, the non-linear element is not involved in the training process. In other words, once one has determined the degree p of the series and the dimensionality D of the input vector, the dependence of the output on the network's parameters is linear, and consequently parameters can be simply calculated, if a MSE cost function is assumed. This happens because the non-linear structure of the network is determined once and for all by D and p. Conversely, in RBFN the non-linearities are parameterised, and therefore we have a non-linear dependence of the output on these parameters. Hence it is impossible, even using the MMSE training, to determine the optimal non-linear parameters by solving a linear system of equations like (2.21). Therefore, non-linear training plays an essential role.

However, being a single-layer network, RBFN benefits from linear techniques in determining the optimal value (in the MMSE sense) of its linear parameter vector $\mathbf{\Phi}$. Together with the simple solution, equation (2.21), for the linear parameter vector, orthogonalisation can be efficiently applied to reduce the number of nodes and the associated complexity of the network.

In this chapter we will cover all these aspects. Firstly, we will briefly recall the problems related to the choice of the aperture (already discussed in the preceding chapter). This will lead us to choose a single aperture for the experiments. Then, we will determine the architectures to be used in the experiments, that have been described in chapter 3.

Secondly, we will describe a suitable procedure to determine the non-linear parameters of a RBFN with Gaussian kernels (i.e. the centres and the width parameters) that does not involve non-linear techniques but rather uses a series of heuristic considerations. This will involve an initial choice of a large number of nodes in the network. In order to obtain a reasonable size for the network, we will successively apply the OLS reduction algorithm. The results of these experiments will be discussed. An algorithm (iterative orthogonalisation) will also be proposed in order to apply OLS to large sets of nodes that generate correlation matrices too large to be reduced in a single application of the OLS algorithm.

Finally, a non-linear optimisation technique, the Nelder-Mead modified simplex algorithm, discussed in detail in appendix C, will be applied to obtain further improvements. The application of this technique to an initially large network pruned via OLS will be compared to the application of the same technique to a small, inefficient, randomly initialised network.

Throughout the results will be compared with those obtained with the linear Wiener filter, as well as the 3rd order Volterra series, both using the same aperture. In order to evaluate the performance in terms of computational complexity, a simple cost analysis outlined in appendix B will be used. The cost comparison will be performed initially with a Volterra series with its full set of nodes. Successively, the series will be pruned using OLS and a more detailed cost comparison will take place.

6.2. Sampling aperture, network size and architecture

From the analysis in the previous chapter we have adopted an 8 pixel aperture (aperture 8_1) for use with a 3rd-order Volterra series. This was based on the evidence that such an aperture produced a reasonable MSE minimisation compared to larger apertures, with a number of nodes (164) that was considered feasible. This choice was made by considering a variety of other apertures. Smaller apertures produce somewhat poorer results, whilst the improvements obtained with larger apertures do not justify the resulting increase of complexity. Furthermore, the size of the series obtained using the aperture 8_1 can be reduced by OLS without excessively degrading the performance.

In order to restrict the number of experiments, and to provide a comparison with Volterra series, the RBFN has been implemented using the aforementioned 8_1 aperture. Clearly, this decision is also justified by the fact that we want to study how the RBFN deals with the same amount of information provided to the Volterra series.

However, as already stated, for a RBFN the number of nodes M is a free parameter. Based on some rough estimates of the computational cost of a RBFN network (appendix B), we deduced that a RBFN network, using the 8_1 aperture, should be built with no more than 10~30 nodes in order to be computationally comparable with the 3^{rd}-order Volterra de-interlacer.

Test will be conducted involving larger networks, which will be successively reduced to the target size by application of the OLS algorithm. This point is very important, because the idea of selecting a large network and successively pruning it by the application of OLS is the basis by which we will try to overcome the need for a non-linear training stage. The following section discusses this approach, and the experimental evidence will provide its justification, as well as showing its limitations.

The final point covered in this section is the choice of the network's architecture. We assume that the kernels are Gaussian. In chapter 3 we have shown different possible architectures that generally go under the guise of RBFN. In this chapter we will consider two architectures. One is a standard RBFN with Gaussian kernels. The other one is the hybrid GRBFN (HGRBFN) that has an explicit linear kernel in parallel to the RBFN core. It will be shown that the hybrid scheme presents superior performance compared to the non-hybrid architecture, especially when the number of nodes is reduced. This superior performance is obtained at a very little increase in computational cost.

6.3. Heuristic choice of centres and determination of width parameter

As already discussed in chapter 3, a possible way to choose a set of centres is to pick them randomly from the input set. If the number of centres M is large enough, this approach leads to a set of centres which mimics the distribution of the input set. Throughout it has been presumed that in a large network the exact functional

expression of the non-linear kernels is not critical to the determination of an effective interpolation. One way to understand this is to imagine a network with a centre for each input. In this case, the functional expression of the kernel may assume the trivial form of the unitary impulse

$$h(\|\mathbf{x}-\mathbf{c}_i\|)=h(r_i)=\begin{cases}1 & r_i=0\\ 0 & r_i\neq 0\end{cases} \tag{6.1}$$

and the linear weight corresponding the *i*-th input is equal to the *i*-th target.

$$\phi_i = t_i \tag{6.2}$$

Clearly such a network completely lacks any form of generalisation and, due to the absence of a width parameter, its input-output mapping is extremely non-smooth. However it is clear that in such hypothesis the problem of finding the non-linear parameters (i.e. centres and widths) is trivial. It seems then likely that, when the number of centres is large, the non-linear problem can be addressed in a simple way. As we will see, this assumption fails dramatically whenever a small number of centres (10~30) is used. Computational problems and the need for sufficient generalisation (i.e. avoiding over-fitted interpolators) suggest that M should be smaller than 1000.

One also need to consider how to determine the width parameter for the Gaussian function. A possible approach was introduced in section 3.5, based on the average distance of each centre from the other centres. This technique is now described in detail. Given the *i*-th centre $\mathbf{c}_i$, the average distance $\bar{\delta}_i$ of $\mathbf{c}_i$ from the remaining centres $\mathbf{c}_j$ is computed

$$\bar{\delta}_i = \frac{1}{M-1}\sum_{j=1}^{M}\|\mathbf{c}_i-\mathbf{c}_j\| \tag{6.3}$$

This mean distance is used to determine the width parameter by specifying the height γ of the Gaussian function at the distance $\bar{\delta}_i$ from the centre $\mathbf{c}_i$

$$\gamma = \frac{h(\bar{\delta}_i)}{\max\limits_{r_i \in R^+} [h(r_i)]} = \frac{h(\bar{\delta}_i)}{h(0)} = h(\bar{\delta}_i) \tag{6.4}$$

We call γ the *overlap factor* since it controls the degree to which the kernels overlap each other, and it follows that $0 < \gamma < 1$. In principle, a different γ can be used for each node; however, this adds to the problem of determining a suitable overlap for all nodes. Thus, a single γ is used for all the centres. Since $h_i(\bar{\delta}_i) = \exp\left(-\bar{\delta}_i^2 / \sigma_i^2\right)$, σ_i^2 is determined using

$$\sigma_i^2 = -\frac{\bar{\delta}_i^{\,2}}{ln(\gamma)} \tag{6.5}$$

Note that fixing γ for all the centres results in different widths for each node. There remains the problem of finding a "good" overlap. The trade-off is between smooth mapping (i.e. broad Gaussians, $\gamma \to 1$) and "crisp" mapping (i.e. narrow Gaussians, $\gamma \to 0$). The two alternatives will be heuristically discussed in the next two sections.

6.4. Random selection of centres

This section presents the results of GRBFN and HGRBFN de-interlacing using the criteria previously considered to select the non-linear parameters. Centres are randomly selected from the input set, and the error is evaluated for 10, 25, 50, 100, 250, 500 and 1000 centres. In order to account the variability of the MSE arising from the random selection process, trials are independently repeated 10 times for each network size, architecture and width factor. The width parameters have been chosen according to the criteria in section 6.3, and overlap factors γ of 0.1 (narrow Gaussian) and 0.9 (broad Gaussian) have been employed.

The results are detailed in figure 1 and tables 1a, b, c and d. The maximum, minimum and mean value of the MSE are reported for each value of M. Note that the results have been compared to those obtained in chapter 5 using the 3rd-order Volterra series

and the linear filter. The curves in figure 1 allow one to understand the general behaviour of the system. Firstly, one can see how random selection progressively fails as the number of centres decreases. The results confirm our previous assumption, since a large number of centres produces a good, consistent performance, with a negligible difference between the maximum and minimum value.

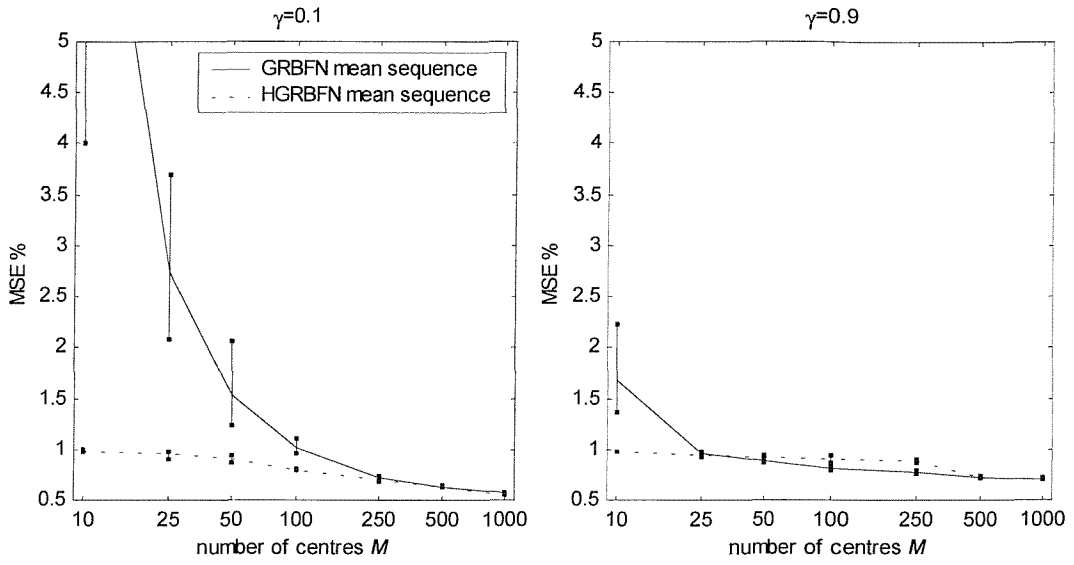

Figure 1. Random selection of centres for GRBFN and HGRBFN, mean values and min-max variation.

The broad Gaussians ($\gamma = 0.9$) help to reduce the MSE degradation for small M, but prevents the network from obtaining good MSE reductions when the number of centres increases. The reason is that as the number of centres increases, a greater local definition is obtained by narrowing the Gaussians ($\gamma = 0.1$). On the other hand, narrow Gaussians fail dramatically when M is small.

These results also indicate that the HGRBFN structure achieves better results than the standard GRBFN when narrow Gaussians are employed. Clearly, the explicit linear path in HGRBFN assures that a reasonable linear de-interlacing is always achieved, even in areas of the input space where the narrow Gaussians fail to produce a good input-target mapping. The difference between the two structures is reduced for the broad Gaussians. This is because the broad Gaussian functions are better able to mimic linear functions with a smaller number of centres. Thus the explicit linear path adds little extra capability.

MSE	NUMBER OF CENTRES M						
	10	25	50	100	250	500	1000
Min	3.9990	2.0805	1.2357	0.9589	0.7029	0.6301	0.5633
Mean	7.9266	2.7452	1.5258	1.0229	0.7202	0.6364	0.5679
Max	17.1336	3.6930	2.0527	1.1112	0.7440	0.6432	0.5730

Table 1.a. GRBFN, randomly initialised, γ=0.1 (Wiener MSE=0.9893. Volterra MSE=0.6956).

	10	25	50	100	250	500	1000
Min	1.3618	0.9270	0.8656	0.7949	0.7588	0.7151	0.7043
Mean	1.6702	0.9578	0.8869	0.8129	0.7736	0.7226	0.7093
Max	2.2181	0.9867	0.9068	0.8388	0.7982	0.7315	0.7151

Table 1.b. GRBFN, randomly initialised, γ=0.9.

	10	25	50	100	250	500	1000
Min	0.9783	0.9036	0.8697	0.7937	0.6868	0.6232	0.5606
Mean	0.9849	0.9563	0.9051	0.8009	0.6997	0.6260	0.5636
Max	0.9882	0.9744	0.9381	0.8125	0.7103	0.6285	0.5673

Table 1.c. Hybrid GRBFN, randomly initialised, γ=0.1.

	10	25	50	100	250	500	1000
Min	0.9725	0.9209	0.8980	0.8738	0.8700	0.7147	0.7029
Mean	0.9798	0.9470	0.9262	0.9037	0.8886	0.7215	0.7081
Max	0.9854	0.9594	0.9472	0.9348	0.9054	0.7299	0.7135

Table 1.d. Hybrid GRBFN, randomly initialised, γ=0.9.

6.5. Orthogonalisation of randomly selected GRBFN and HGRBFN

In the previous section, it has been shown how the heuristic approach of choosing the centres randomly from the input set and selecting the width parameter on the basis of a general overlapping criterion fails when M is small. However, this selection criterion is a preliminary step towards a more structured technique. Since a large, randomly selected network succeeds in producing a good mapping, it is reasonable to assume that a large number of nodes, with a proper choice of centres and width

parameter, contains most of the necessary information in order to produce a satisfactory interpolation. We will now show the results of applying the OLS algorithm described in chapter 4, in an attempt to extract this information from an initially large network.

There are several reasons that justify the application of this selection-reduction technique. Standard non-linear methods are not guaranteed to converge to a global optimum, and may be strongly dependent on the initial conditions. Moreover, non-linear optimisation techniques usually require a large number of iterations. Since we assume that a random selection of 1000 centres contains (in principle at least) a sufficient information to solve the problem, we might wish to extract as much of this information as possible from the surrounding haze of redundancy, rather than surrender to a "brute-force" approach. Further non-linear optimisation can be used at a later stage.

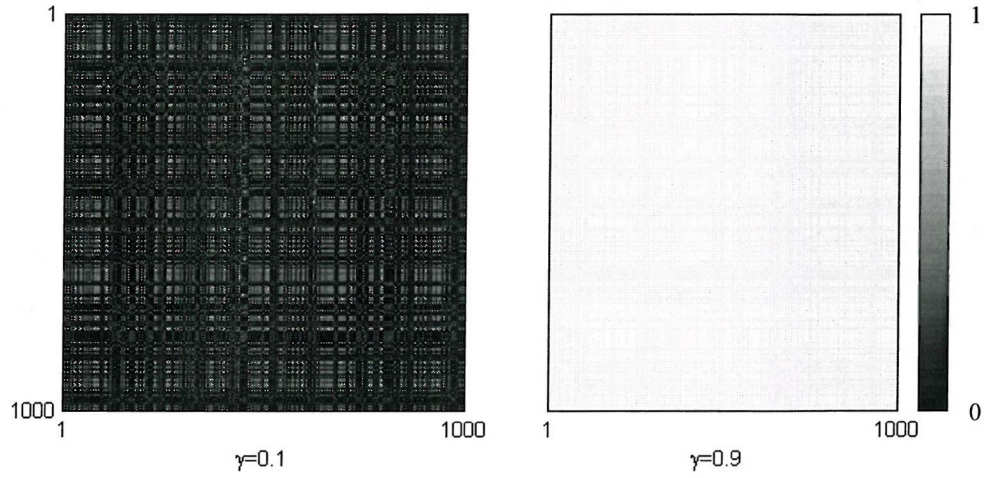

Figure 2. Normalised correlation matrix of RBFN regressors. γ=0.1 (left) and γ=0.9 (Right).

If we plot the normalised cross-correlation matrix **R** for 1000 centres, γ=0.1 and γ=0.9 with a GRBFN architecture, we can clearly see that there is linear redundancy, since the regressors are highly correlated (Figure 2). Note that regressors with γ=0.9 are more correlated than those with γ=0.1, as portrayed by the brighter shades reflecting the greater correlation. This is a consequence of the broad support which means that many of the regressors repeat similar information, since they overlap the same (broad)

region of the input space. Figure 2 gives a visual explanation as to why the use of a large number of broad regressors does not produce the reduction in MSE obtained with narrow regressors. However figure 2 also explains why broad Gaussians perform better with a smaller number of regressors.

Tables 2a, b show the MSE obtained using the OLS reduction for γ=0.05, 0.1, 0.5, 0.9. The networks for which the MSE is smaller than that obtained using the Volterra series are shaded in light grey, and the MSE which are worse than the Wiener filter are shaded in dark grey. Not surprisingly, hybrid architectures produce MSE that are always smaller than the Wiener MSE. Figures 3 and 4 show the plot of these sequences.

γ	CENTRES						
	10	25	50	100	250	500	1000
0.05	1.9423	1.2072	0.9320	0.7824	0.6651	0.5972	0.5695
0.1	1.8226	1.1071	0.8534	0.7374	0.6387	0.5872	0.5658
0.5	1.1602	0.8659	0.7557	0.6965	0.6414	0.6231	0.6125
0.9	0.9936	0.9423	0.8320	0.7598	0.7306	0.7174	0.7080

Table 2.a. OLS GRBFN, (Wiener MSE=0.9893. Volterra MSE=0.6956).

γ	CENTRES						
	10	25	50	100	250	500	1000
0.05	0.9377	0.8279	0.7793	0.7199	0.6493	0.5890	0.5593
0.1	0.9528	0.8534	0.7765	0.7142	0.6376	0.5897	0.5616
0.5	0.9839	0.8726	0.7563	0.7009	0.6404	0.6237	0.6113
0.9	0.9816	0.8947	0.7981	0.7515	0.7298	0.7221	0.7058

Table 2.b. OLS HGRBFN.

Figure 5.a and 5.b provide a visual comparison between different de-interlaced frames. A hybrid network with 100 OLS reduced centres leads to a result that is visually comparable to the result of a Volterra series. However, the computational cost of such a network is too high to make it a convenient solution (see appendix B). A network with 10 OLS centres has a computational cost that is comparable with that

of a Volterra series, however its performance is only just better that the linear case. This highlights the need to include some non-linear optimisation stage in our training strategy.

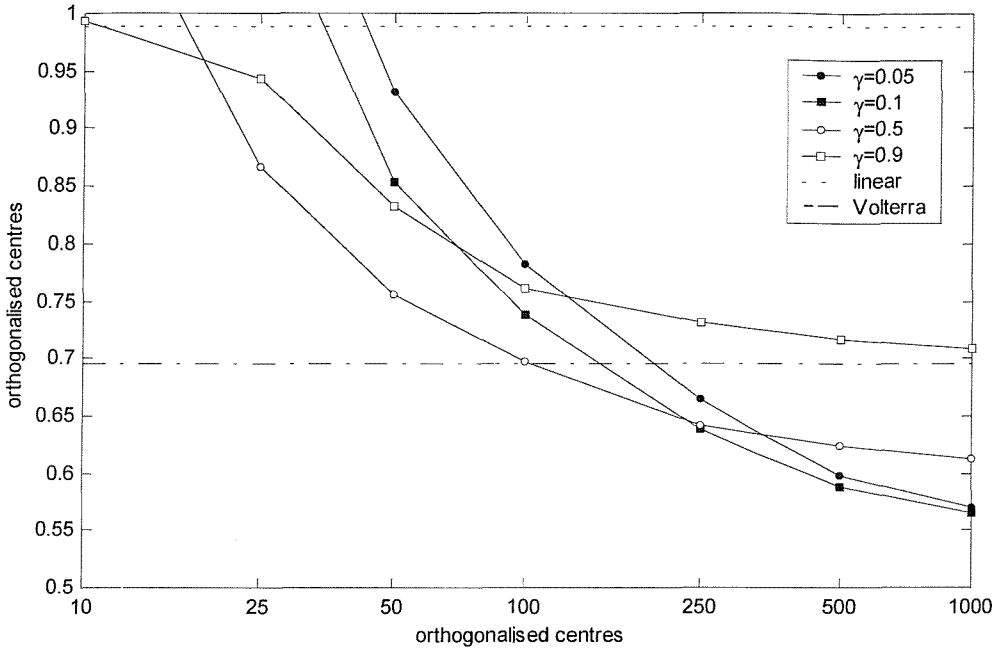

Figure 3. OLS sequence, GRBFN.

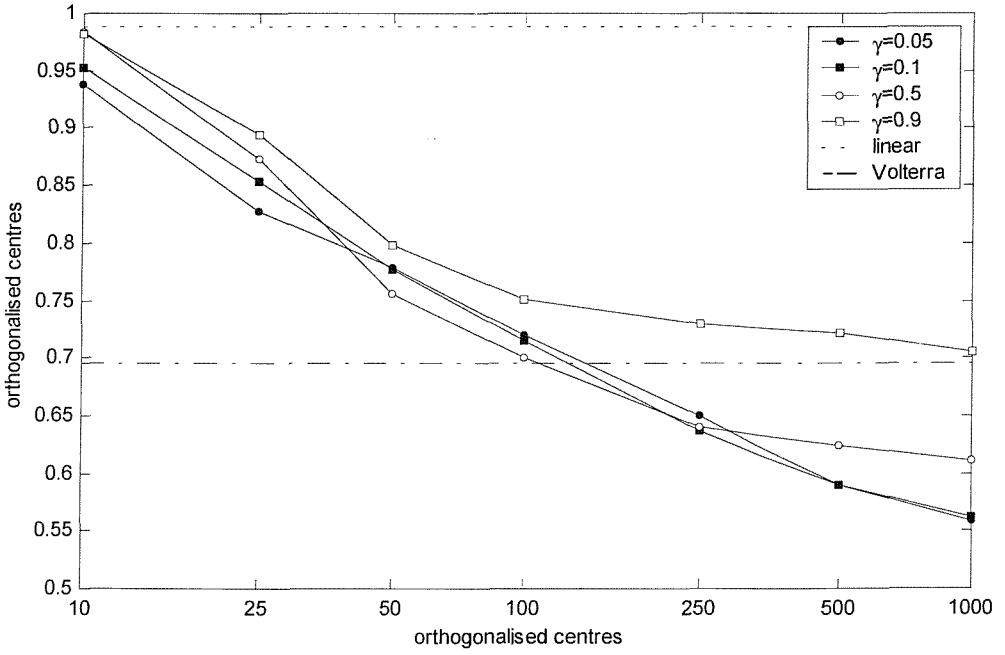

Figure 4. OLS sequence, HGRBFN.

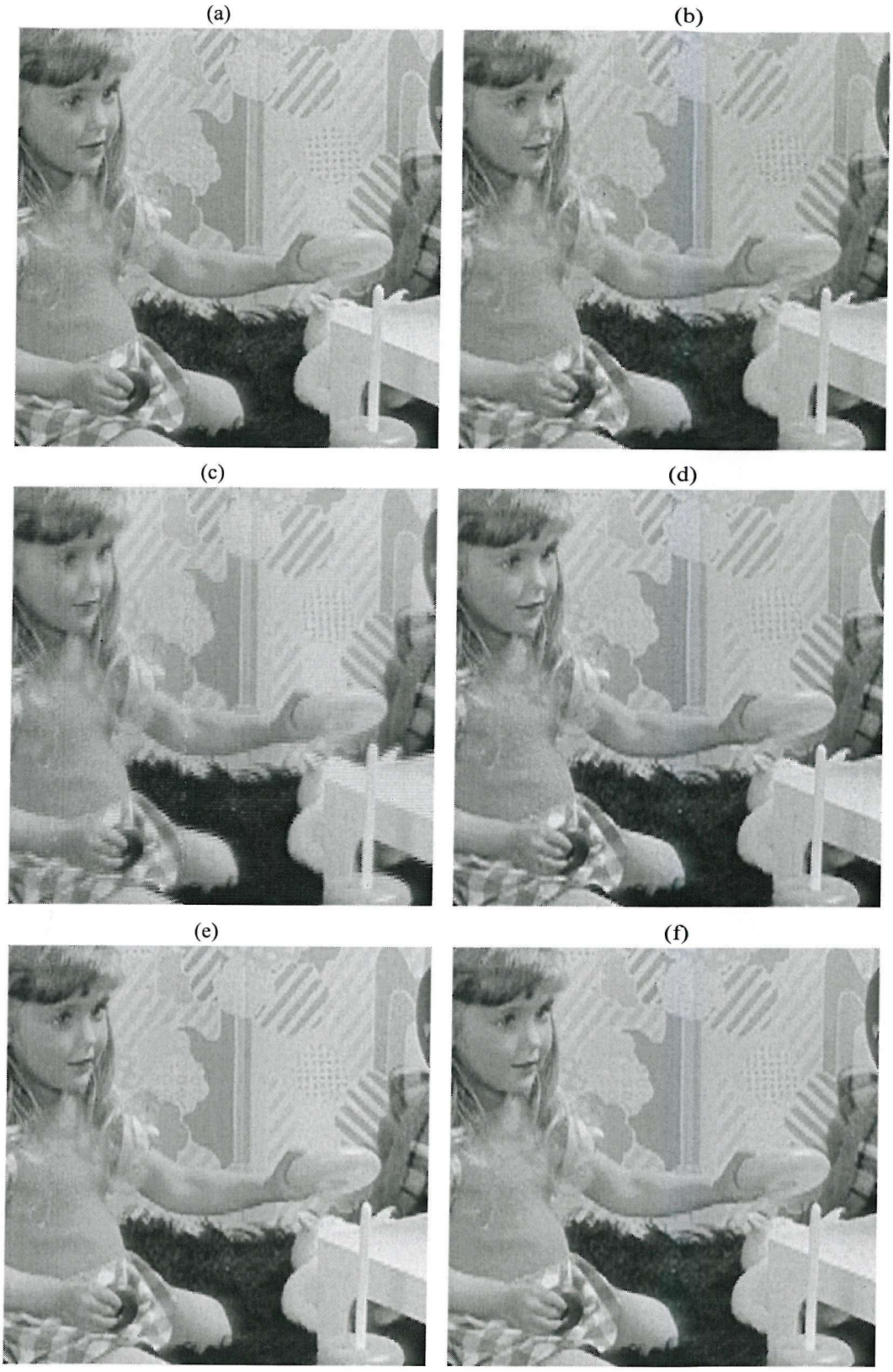

Figure 5.a. (a) source frame, (b) Linear interpolation, (c) random RBFN 10 centres, (d) OLS Hybrid RBFN 10 centres, (e) OLS Hybrid RBFN 100 centres, (f) 3rd order Volterra series.

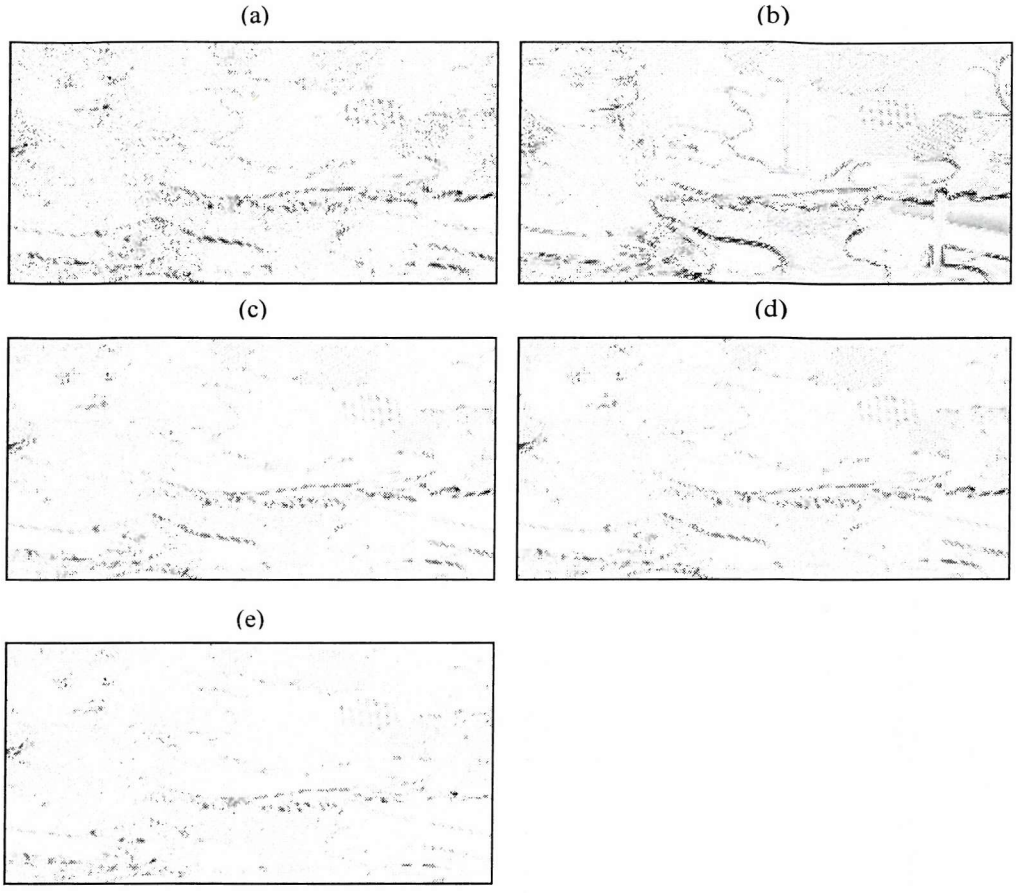

Figure 5.b. Difference fields with respects to the target field. (a) Linear interpolation, (b) random RBFN 10 centres, (c) OLS Hybrid RBFN 10 centres, (d) OLS Hybrid RBFN 100 centres, (e) 3rd order Volterra series.

6.6. Iterative orthogonalisation

The experiments in the previous section illustrate that OLS has the ability to select medium-sized sub-optimal sets of nodes from an initially large population. The size M of this initial set is a trade-off between the ability to represent the training set, and OLS computational demands that increase as the initial set broadens. In section 6.3 we have seen how different sets of 1000 centres, randomly chosen from the input set give rise to similar MSE. It could be assumed that a subsequent OLS reduction to M' nodes shows the same stability, regardless of the 1000 centres initially selected. In fact, when M' is small compared to M (for instance, M'=10), two different initial sets of M regressors may result in two very different MSE when successively reduced to M'. In figure 6, the average results of 10 OLS reductions from 1000 randomly selected centres are plotted together with their maximum and minimum value.

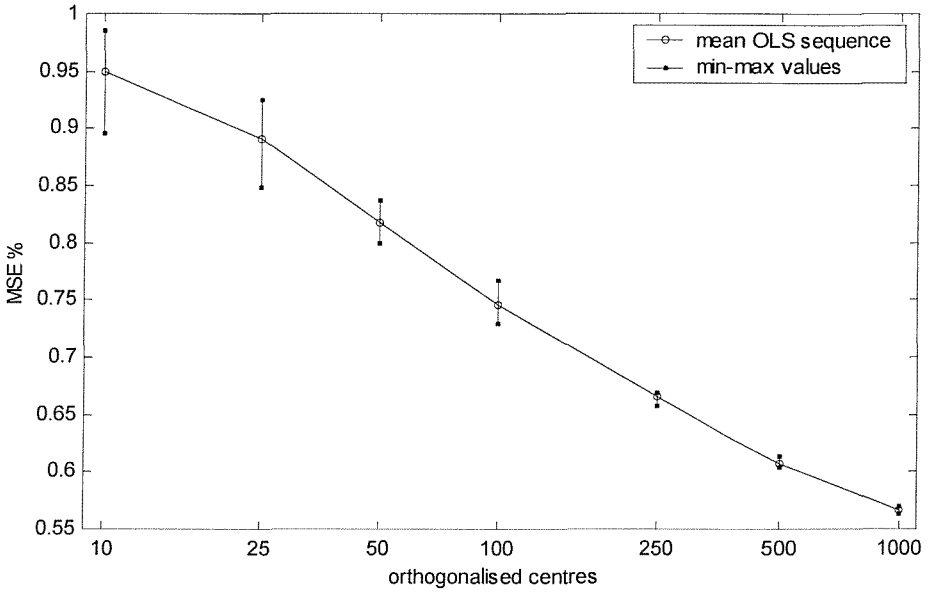

Figure 6. Variability of OLS reduction over 10 different initial set of 1000 regressors.

It is clear that, as the reduction increases, the variability also increases. There are a large number of initial sets of M vectors in the input space that produce an initial good result, but only a small number of these produce a satisfactory reduction. In other words, OLS successfully removes the redundancy from a large set of centres, extracting the relevant regressors. However a given set, randomly selected from the input space, may not contain the optimal M' centres. It seems reasonable to assume

that, the smaller M' we want to achieve, the larger the initial set should be in order to increase the chance to include the "best" M' regressors set. Unfortunately this assumption leads to computational problems, since OLS should operate on correlation matrices as large as, in principle, the whole input set.

This section describes a sub-optimal strategy to compare and orthogonalise large number of candidates, keeping M small at each orthogonalisation phase. The principle is to perform separate orthogonalisations of M' regressors from each of G groups of M candidates, followed by randomly rearranging the $M' \cdot G$ outcomes into new groups of M candidates and repeating the orthogonalisation. A progressive reduction of the number of groups leads to a final orthogonalisation of M' centres from M candidates. This algorithm is depicted in figure 7, with M=1000, and M'=10, 25, 50 and 100.

The results are shown in figure 8 and table 3. As we can see, the algorithm is able to extract progressively better sets of regressors. For $M'>$ 25, the final outcome is equivalent to the best of the initial groups of candidates. However for smaller number of centres this ability fades. In this case, iterative orthogonalisation tends to approach the average output value of the initial sets. This algorithm has not been fully assessed, however it provides additional evidence for the assumption that large numbers of randomly selected centres that may lead to similar results are not guaranteed to yield equivalent results once they are reduced via OLS. By choosing the initial set in a more appropriate manner than just randomly selecting the centres from the input set, one can obtain better reductions. It could be wiser to apply iterative schemes like the algorithm proposed here in order to create large number of centres that yield sub-optimal results once they are reduced.

A possible approach, which has similarities with the method discussed here, could be the use of genetic algorithms (Goldberg, 1953). However this approach will not be considered in this work. Work in this field include Coley (1999), and Mitchell (1998).

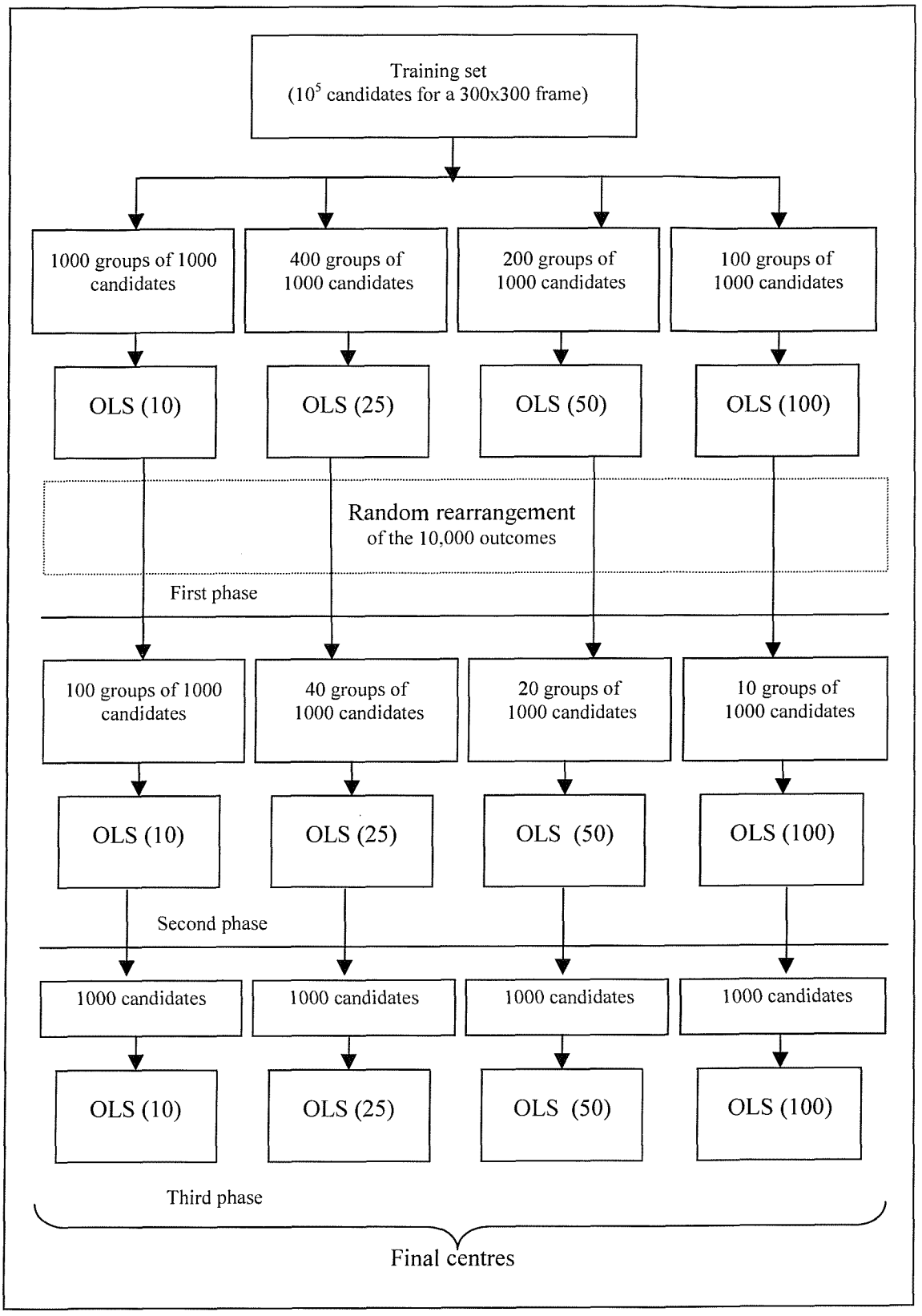

Figure 7. Iterative orthogonalisation algorithm.

			phase1	phase2	phase3
target centres	10	min	0.8568	0.9343	
		mean	0.9586	0.9645	0.9559
		max	0.9861	0.9822	
	25	min	0.8240	0.8109	
		mean	0.8774	0.8409	0.8240
		max	0.9562	0.8759	
	50	min	0.7589	0.7415	
		mean	0.7883	0.7491	0.7325
		max	0.8508	0.7598	
	100	min	0.7034	0.6787	
		mean	0.7206	0.6937	0.6803
		max	0.7360	0.7025	

Table 3. Iterative orthogonalisation: comparison of the results at different phases.

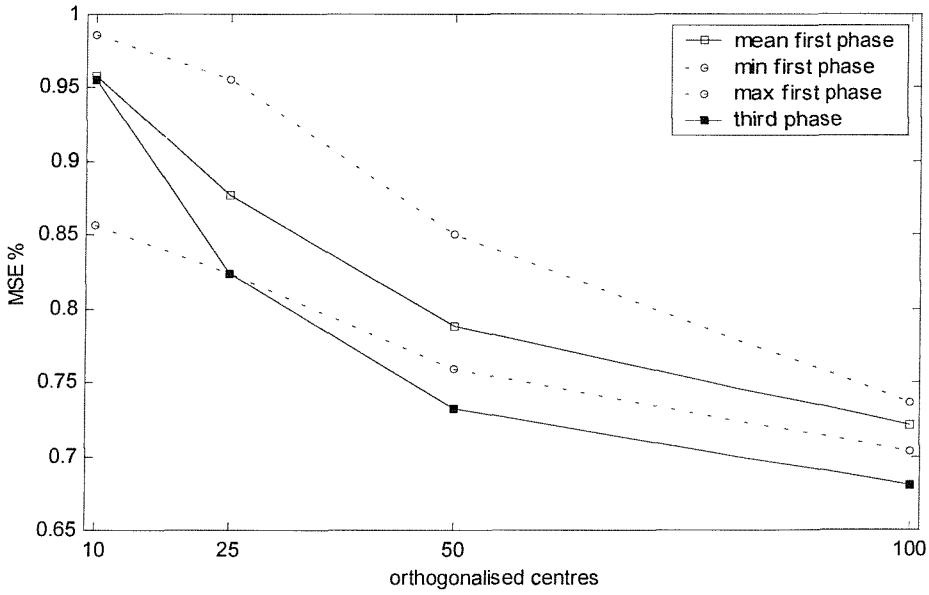

Figure 8. Iterative orthogonalisation, results of the first and third phases.

6.7. Non-linear optimisation

Previous sections have demonstrated the limitations of linear training in RBFN. We have tried to overcome the necessity of non-linear training of centres and widths by choosing large networks, whose size is reduced using the OLS algorithm. In this approach, centres are selected randomly from the input set, and widths are calculated using a heuristic approach. The quantities left to be determined are the linear weights, which can be easily found using standard linear techniques. This approach fails to yield good results as the number of centres falls below a certain value.

It is necessary to apply some form of non-linear optimisation for the centres and widths. This section considers the use of a well-known optimisation technique, the Nelder-Mead (NM) modified simplex algorithm (Nelder, Mead 1965). The algorithm is described in appendix C. Other techniques, e.g. the Gradient Descent (GD) algorithm (Fletcher 1987), are also applicable. Differently from the GD algorithm, NM does not require knowledge of the gradient of the function.

Firstly, OLS is applied to a HRBFN with γ=0.1 to produce two sets of 10 and 25 centres from 1000 candidates. These two OLS networks are trained using the NM algorithm. The results are compared with two HGRBFN initialised with a random selection of 10 and 25 centres. The aim is to see if these two networks converge, i.e. to explore whether the OLS reduction offers an effective technique to create better initialisation for NM training. The number of iterations of the NM algorithm is fixed at 800 times the number of parameters.

The results are shown in figure 9, and in table 4. From these results one can see that OLS creates good initial condition for the non-linear techniques. Note that, with 25 centres, orthogonalisation followed by optimisation achieves a performance equivalent to that of a randomly initialised network, optimised with NM, in about half the number of iterations. For 10 centres however, this advantage is almost completely lost. The curves are similar and there is little advantage in applying OLS before the NM algorithm. It seems that below a certain threshold the random selection strategy, outlined in sections 6.2 and 6.3, produces a poor network. A non-linear technique like NM recalculates the networks parameters to make them fit the problem. In figure 9

the results have also been compared to those obtained by the simple use of OLS (horizontal dashed lines), and the optimised 10 centre network achieves a performance equivalent to a 50 centre OLS network. Whereas the 25 centre network performance improves to a level comparable with a 75 centre OLS network.

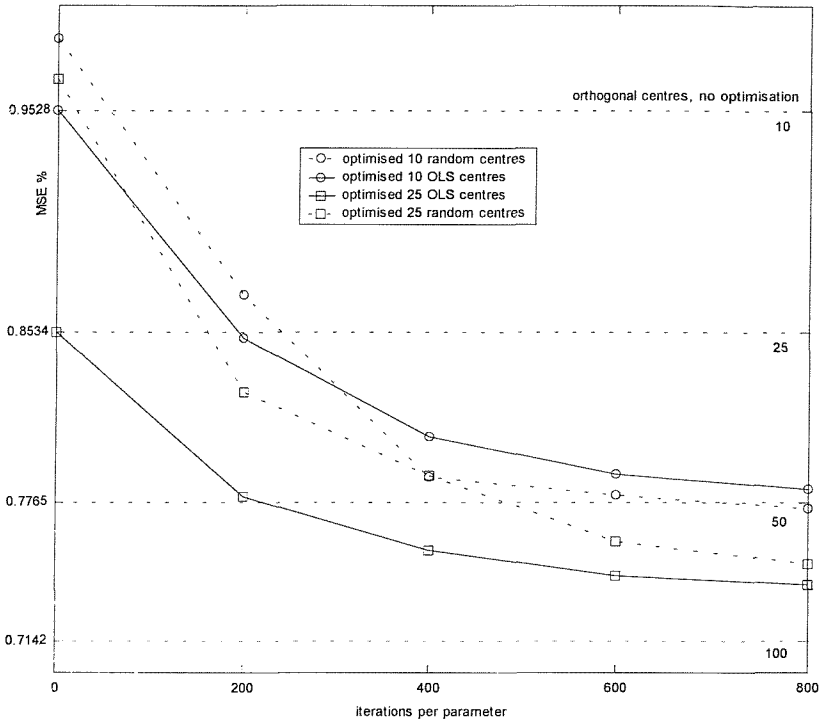

Figure 9. Nelder Mead optimisation of random RBFN and OLS RBFN.

	Iterations per parameter				
	Initial value	200	400	600	800
Random 10	0.9850	0.8703	0.7880	0.7799	0.7737
OLS 10	0.9528	0.8508	0.8060	0.7898	0.7825
Random 25	0.9673	0.8266	0.7886	0.7590	0.7491
OLS 25	0.8534	0.7796	0.7554	0.7434	0.7397

Table 4. Nelder-Mead optimisation of randomly initialised and OLS HGRBFN (Volterra MSE=0.6956)

6.8. Comparison between HGRBN and OLS Volterra series

In previous sections the performance of the HGRBFN has been compared with the performance of the 3rd-order Volterra series. From the analysis in appendix B the computational cost of a HGRBFN can be compared to the cost of the Volterra series if its number of centres is smaller than 30. Experiments suggest that the best HGRBFN performance in this range (table 4, 25 OLS centres), obtained by the combined use of OLS and NM, is slightly worse than the performance of a Volterra series. However there is little qualitative difference in the results, and the two techniques are roughly equivalent in terms of cost and performance.

In order to produce a fairer comparison, one should compare the orthogonalised HGRBFN with an orthogonalised Volterra series. In chapter 5 it was shown that the Volterra series benefits significantly from OLS. Recall that the Volterra nodes have differing costs, according to the degree of their monomials, so the reduction in cost is not necessarily in proportion to the reduction in the number of nodes. For example we know that, although the first four Volterra regressors are linear, the succeeding ones are mainly cubic and therefore impose a higher computational cost.

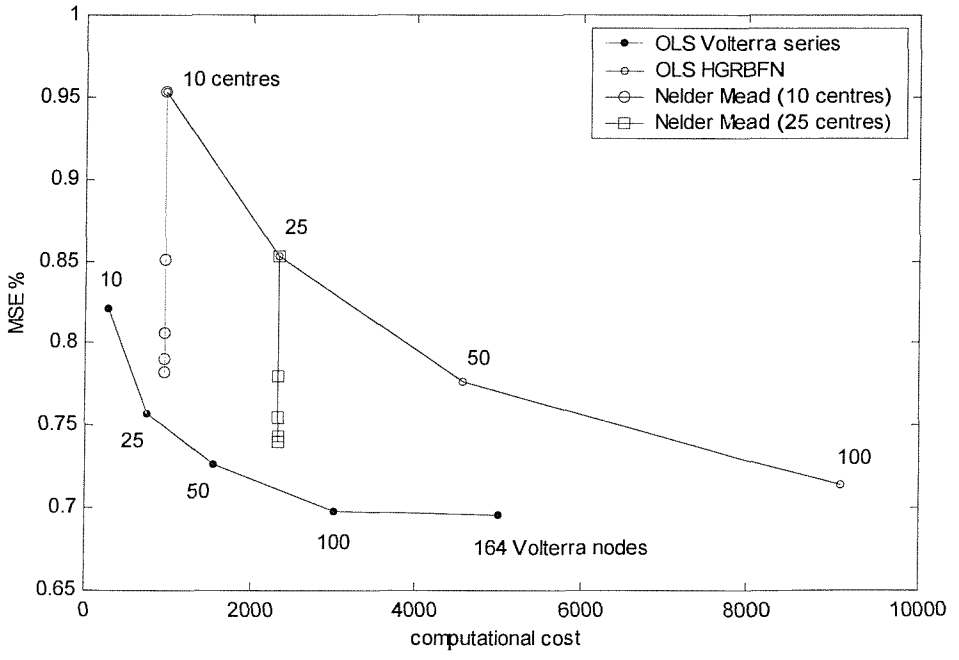

Figure 10. MSE vs. cost plot, OLS-NM HGRBFN, OLS 3rd-order Volterra series.

Figure 10 shows the MSE-versus-cost plot for the 10 and 25 centre OLS-NM HGRBFN networks, and the OLS Volterra series. This is based on assuming that multiplication is eight times the cost of a sum, $C_{mult} = 8 \cdot C_{sum}$ (see appendix B). This assumption tends to favour the HRBFN (given the high number of products in Volterra series) and the break-even network size is 50 centres rather than 30.

6.9. Representation of the input space

This section presents some tools to aid the understanding of the complex phenomena arising in these experiments. Given the strong topological nature of RBFN, it is useful to have some form of graphical representation of the input space and of the centre set, that by construction belong to the same space. This representation is particularly useful when localising kernels, such as Gaussians, are used, allowing one to consider the "area of influence" of a kernel, i.e. the local neighbourhood of the corresponding centre. Unfortunately, as the dimensionality of the input set D is bigger than 3, there is no natural representation of the input space.

Nevertheless, it is useful to represent the input set in some projected planes of the sampling lattice, i.e. projecting $D-2$ dimensions into a plane. Figure 11 shows examples of the projected inputs for the 8_1-aperture. Note that only 6 out of 32 possible projected planes are shown. This choice is intentional, and will be justified later in chapter 9, by considering the symmetry properties of images.

From figure 11 one can see that there is a trend for inputs to cluster around the main diagonal of the input space, in a sort of elliptical "bubble". Of course, this trend might be misleading, since the full 8-dimensional manifold could be far more complicated. Note that the projections based on neighbouring pixels, e.g. x_3 and x_6, are more concentrated on the diagonal than projections based on distant pixels, e.g. x_1-x_8. This reflects the higher correlation between adjacent pixels.

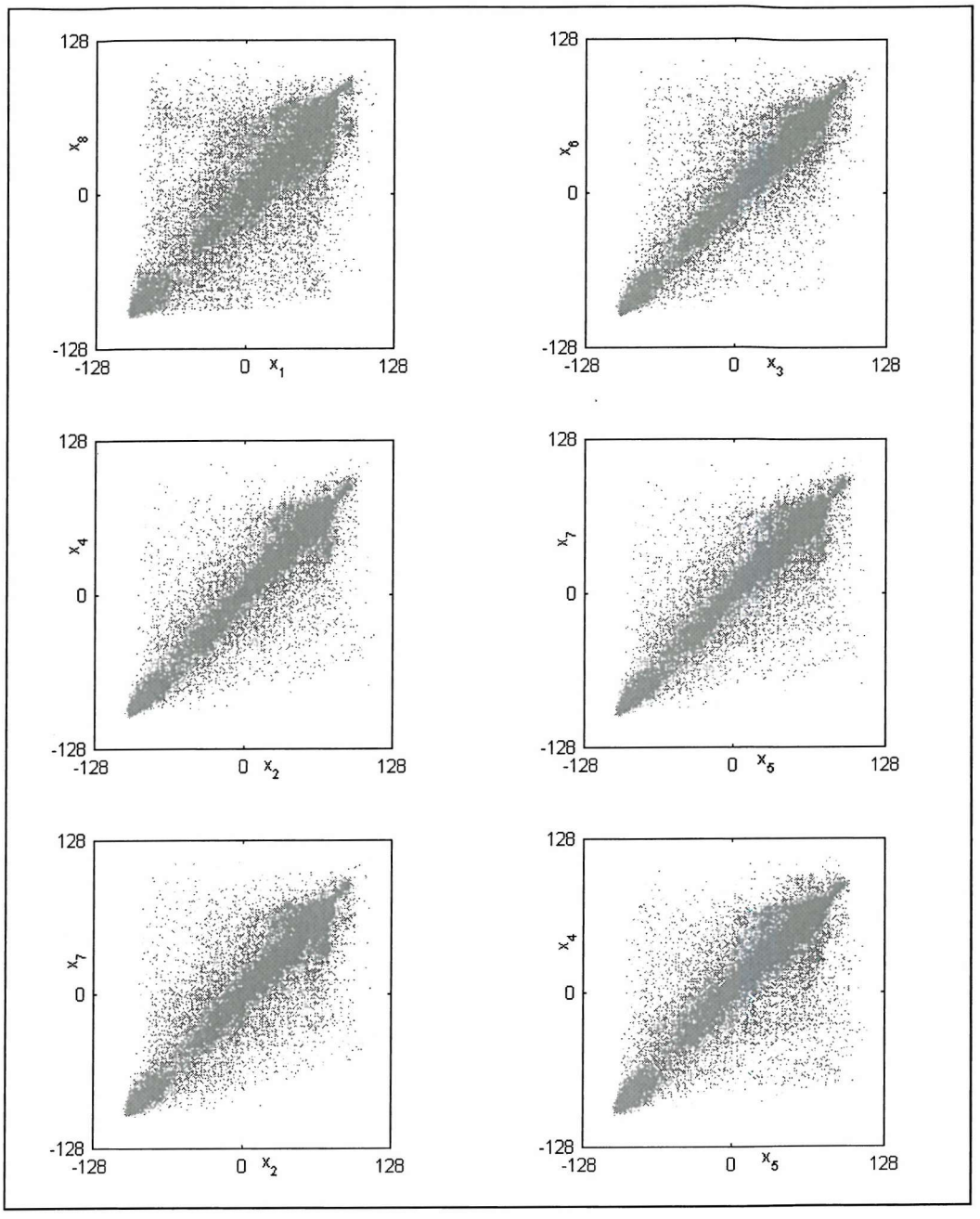

Figure 11. Input space in some of the projected planes.

Simple considerations of the picture make this conjectured clustering reasonable. An input vector on the main diagonal is simply a vector arising from the sampling of a uniform area of the image. In fact, in many images there is a reasonable prevalence of uniform or quasi-uniform areas. One must take care however not to consider this as a property of all images. As a counter-example, a picture of a "shaded chessboard"

generates inputs concentrated on both diagonals of an 8-dimensional cube, as illustrated in figure 12.

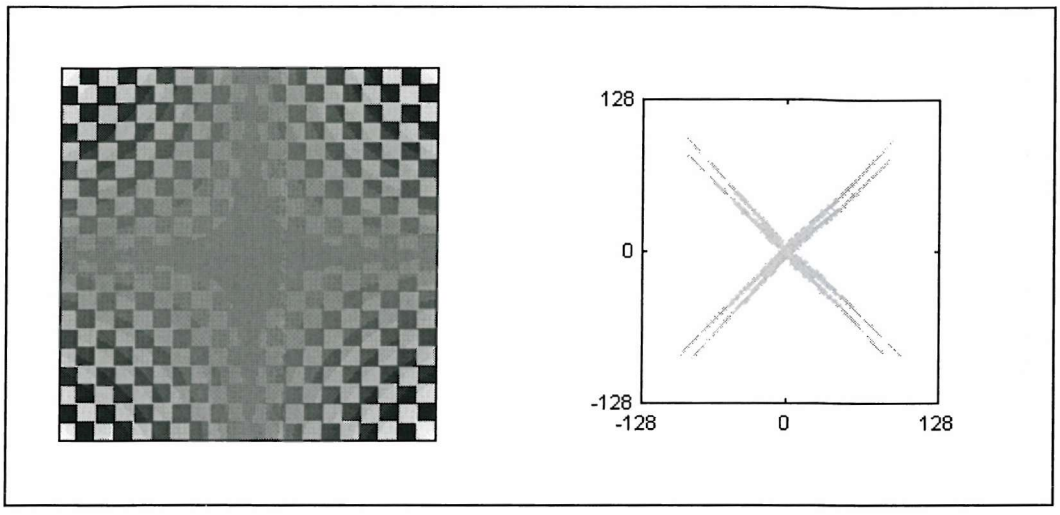

Figure 12. "shaded chessboard" input distribution in the projected plane x_1-x_8.

6.9.1. Orthogonal centres

As an example figure 13 illustrates two projected planes for a sequence of OLS centres of a HGRBFN with width parameter $\gamma = 0.1$. The projected planes corresponding to the external (x_1-x_8) and internal (x_3-x_6) vertical taps of the sampling lattice are shown. Remember that in the linear case, these taps carry most of the information used by the linear filter, hence they are likely to be of greatest importance even in the non-linear case. Note how the centres move apart as the number of centres is decreased. This is the effect of orthogonalisation removing overlapping centres.

6.9.2. Nelder-Mead centres

In this section, the behaviour of the centres in a hybrid Gaussian RBFN optimised with the NM algorithm are discussed. Figure 14 shows the results for a set of 25 orthogonal centres. The most important characteristic is that some centres move a long way outside the input cluster. It seems that to achieve better performance one should move some centres outside the input cluster. It is reasonable to consider that the outlying centres generate smooth, quasi-uniform slopes over the input cluster,

since as $r = \|\mathbf{x} - \mathbf{c}\|$ becomes large, the exponential function becomes more like a linear plane over the input space.

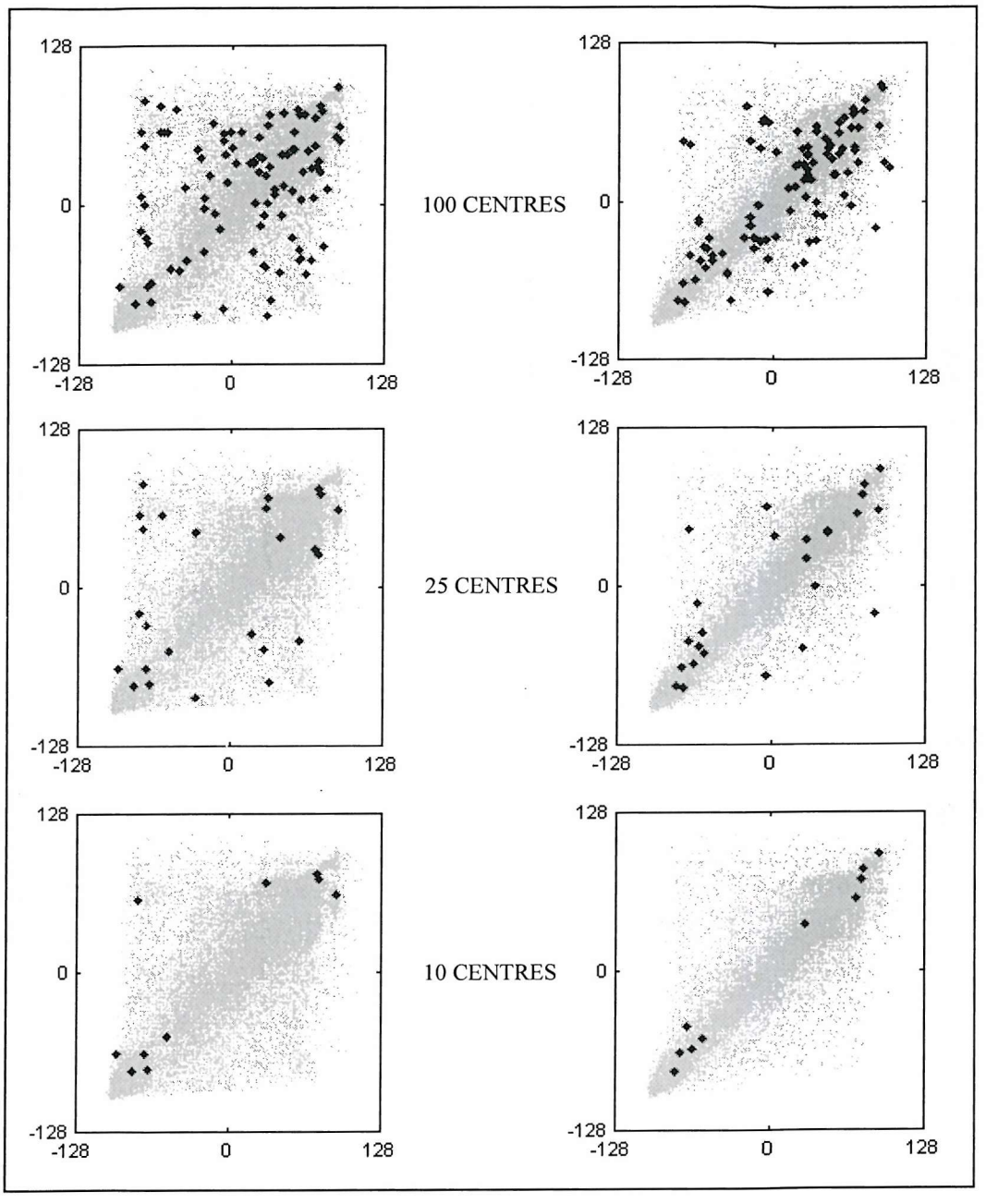

Figure 13. OLS centres (black diamonds) in the projected planes x_1-x_8 (left) and x_3-x_6 (right).

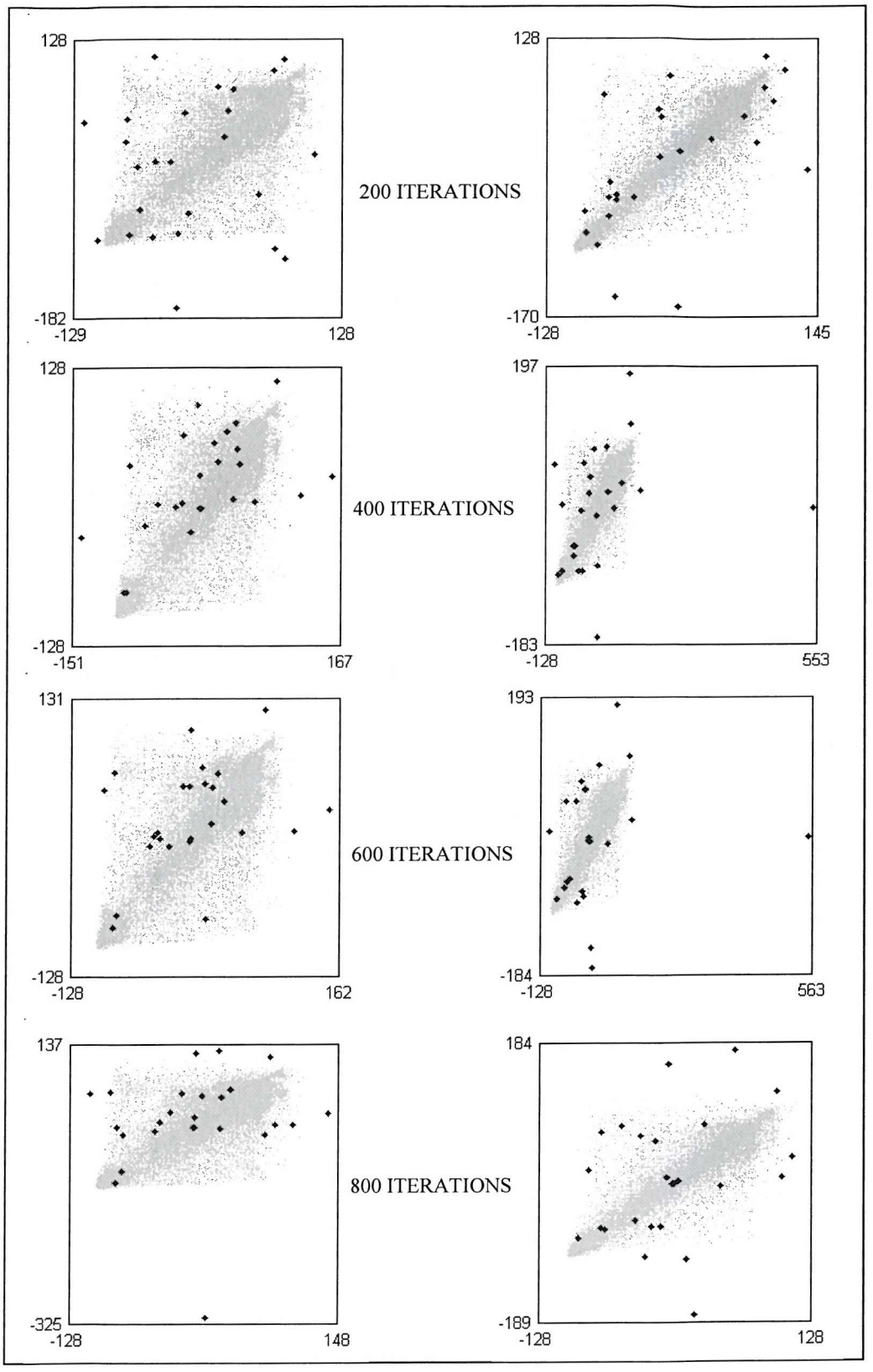

Figure 14. NM optimisation, orthogonal hybrid GRBFN. planes x_1-x_8 (left), x_3-x_6 (right).

6.10. Conclusions

In this chapter the application of RBFN in de-interlacing has been investigated. The main task was to construct an efficient RBFN with performance and computational cost comparable to a Volterra de-interlacer. A hybrid RBFN employing Gaussian Basis Functions acting on the 8_1 aperture was selected.

The training of a RBFN requires the optimisation of its non-linear parameters, the centres and the widths. In order to avoid the difficulties involved in optimising those parameters, a different approach has been chosen. It has been shown that large networks are relatively resilient to the choice of the non-linear parameters. Specifically, a random selection of centres, followed by a heuristic determination of the widths, lead to good results if the network has a relatively large number of nodes. The application of OLS to prune the network of redundant branches has proven successful to reduce the size of the network by a significant degree. The iterative application of the OLS algorithm has proven successful to reduce the variability of the result given by the initial conditions.

However, the computational constraints of the problem impose a target size of the network that cannot be reached by the simple linear OLS training without degrading the performance. The application of a simple iterative optimisation technique to the centres and widths of the network proves successful to achieve a good result using the target number of centres.

The main crux of the application of RBFN in de-interlacing is the reduction of the number of nodes. In chapter 8 some constraints that a de-interlacing system must satisfy will be discussed. These constraints, applied the RBFN, will lead to a further reduction of the number of centres, as discussed in chapter 9.

The two non-linear techniques investigated, the Volterra series and the RBFN, produce results that are substantially equivalent, in terms of interpolation error and computational cost. However, the investigation has been carried on a single test frame. The next chapter discusses the performance on a wider set of images.

7. GENERALISATION

7.1. Introduction

In previous chapters, the ability of linear and non-linear techniques to produce reliable de-interlacing systems has been examined. Wiener linear filters, Volterra series and RBFN have been used to de-interlace a sample frame ("girl") and the results have been discussed. The main goal of these experiments has been to minimise the interpolation error, in the MSE sense. Obviously, a realistic de-interlacing system must produce satisfactory results over a range of inputs. However, real images have very different characteristics from each other. It is questionable then how a system, trained on a single image, can produce satisfactory results on a range of other inputs.

As previously emphasised, the goal of a training procedure is not to learn the exact pattern presented as the training set, but rather to create a statistical model of the process that generates the training data. This is of paramount importance if the network is meant to produce satisfactory results when interpolating unknown patterns. From this viewpoint, the training set is just a particular realisation of a more general process.

The ability of a model to predict values not seen in the training set is called its *generalisation*. The most intuitive way of achieving generalisation is to extend the training set to include as many training patterns as possible. Clearly any realistic training procedure limits the size of the training set, and even a set made of a modest number of frames might prove infeasible for high-speed computers. More generally, it is often impossible to produce a large training set, as sometimes there might be scarcity of available data. But there is a more subtle reason that leads us to pursue generalisation using a limited number of training patterns. In fact, one wish to build a system with a-priori generalisation abilities by creating appropriate training procedures, rather than surrender to a "brute-force" approach.

This chapter addresses the problem of generalisation firstly introducing a larger general set of frames, which will be used either as training set or as test set to assess the generalisation performance. Prior to presenting the results, two key parameters in assessing the generalisation ability of a network, the *bias* and *variance*, will be discussed.

The next step will be to use a generalisation technique known as *weight decay* (Bishop, 1995). This technique draws its theoretical background from the theory of regularisation. An extended training set will be used in order to enhance the system ability to generalise. This extended set deliberately includes one frame that is very different from the others. Whilst most of the frames represent "natural" scenes, this special frame is a page of text.

This investigation also considers the possibility of adding a-priori knowledge to the training procedure. For some specific patterns we can determine, a-priori, the desired output. It is desirable that our training is constrained to abide to these known patterns, that the unconstrained MMSE solution does not generally achieve. This introduces the subject of *constrained optimisation*, and it will be seen how this affects the performance.

Finally, a *mixture of experts* (Jacobs *et al.*, 1991) is introduced that overcomes many of the limitations of alternative methods. Whenever a single, fixed-parameters network does not deliver satisfactory results, it may be convenient to adopt a more flexible solution, where the parameters are changed adaptively according to the current input. This approach can be seen as a pool of networks that specialise their mapping on different clusters of inputs. A decision algorithm determines which network to use according to some features in the input. It is clear that the main drawback of this approach is the increased computational complexity. Therefore it is necessary to maximise the performance with the minimum additional cost.

7.2. The frame set

Figures 1.a to 1.f show the frames that will be used in the following experiments. Figures 2.a to 2.f show their normalised histograms. One can see that the images present different visual features, like different textures, borders, uniform areas, etc. The different shapes of the histograms in part reflect this variety. In particular, frames 4 and 6 have peculiar characteristics that differentiate them from the other frames.

Frame 4 has a characteristic line texture, appearing in the oblique rows in the man's shirt, that represent an insolvable problem to many de-interlacing systems, since the corresponding spectral contribution is located in the high-frequency part of the spectrum. Specifically, in some regions the spacing between a dark line and a bright line will be exactly one pixel in the vertical direction. In the resulting source field, a uniform bright area needs to be replaced by a series of alternating bright and dark lines. This is clearly a case of unrecoverable aliasing, since the sampling process has completely destroyed the original information. Figures 3.a and 3.b show the frequency spectra for frames 3 and 4 respectively. The broader frequency content of frame 4 is evident. Also figures 3.c and 3.d show the projected distributions for these two frames. The greater spread in figure 3.d arises because of the fact that dark and light lines are in close proximity.

Frame 6 ("latin") is of particular interest, since it differs in many ways from the "natural" scenes depicted in the other frames. The corresponding patterns are mainly uniform black areas (background), uniform white areas (the letters) and sharp transition areas. One can see from figures 4.a and 4.b that the spectrum of frame 6 has a broader frequency content compared to frame 3, and the distribution of inputs is rather different, being mainly concentrated on the borders of the input space. It will be shown how this will dramatically affect the OLS training of RBFN.

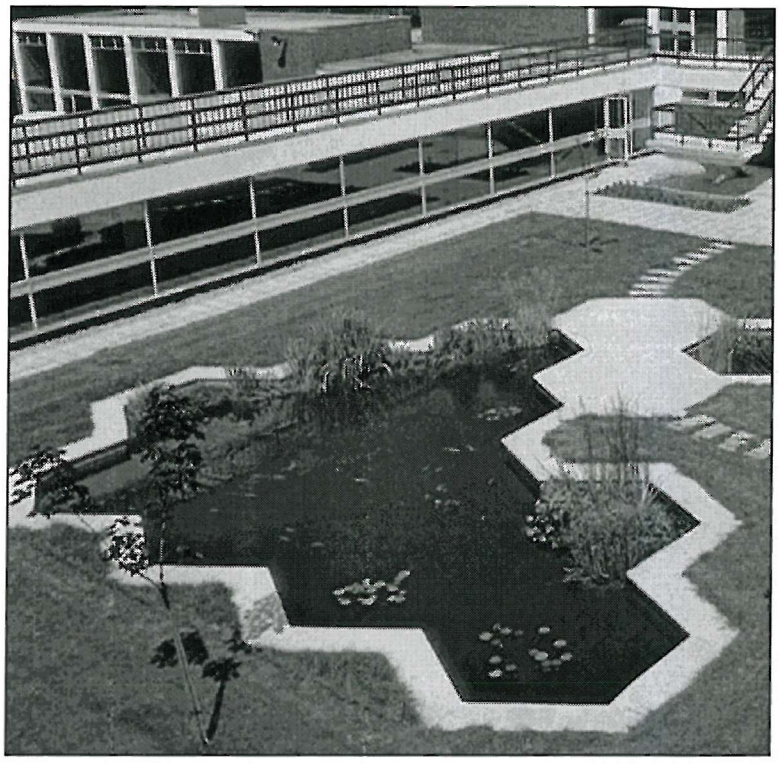

Figure 1.a. Frame 1 ("pond").

Figure 1.b. Frame 2 ("boat").

Figure 1.c. Frame 3 ("girl").

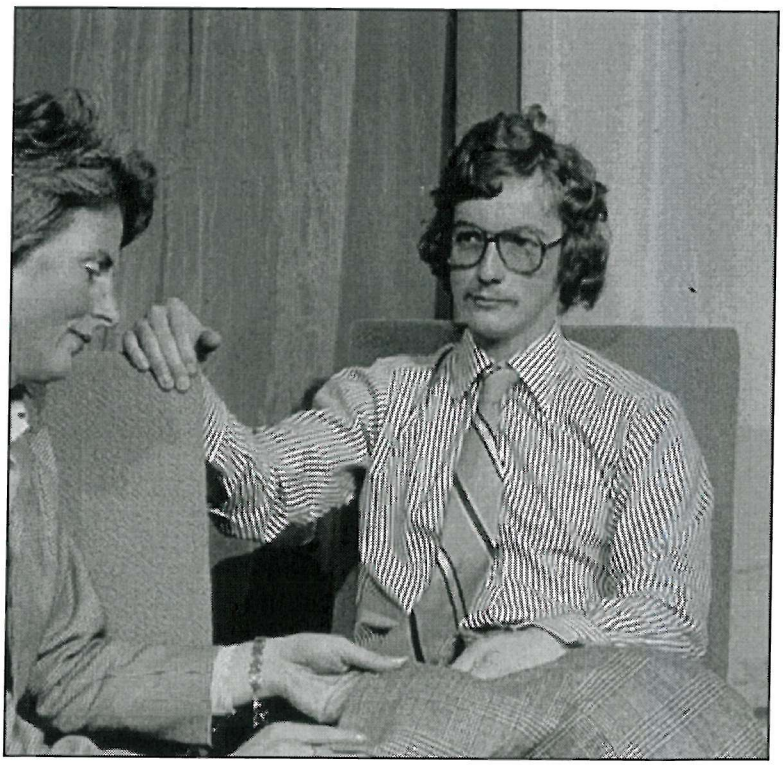

Figure 1.d. Frame 4 ("shirt").

Figure 1.e. Frame 5 ("face").

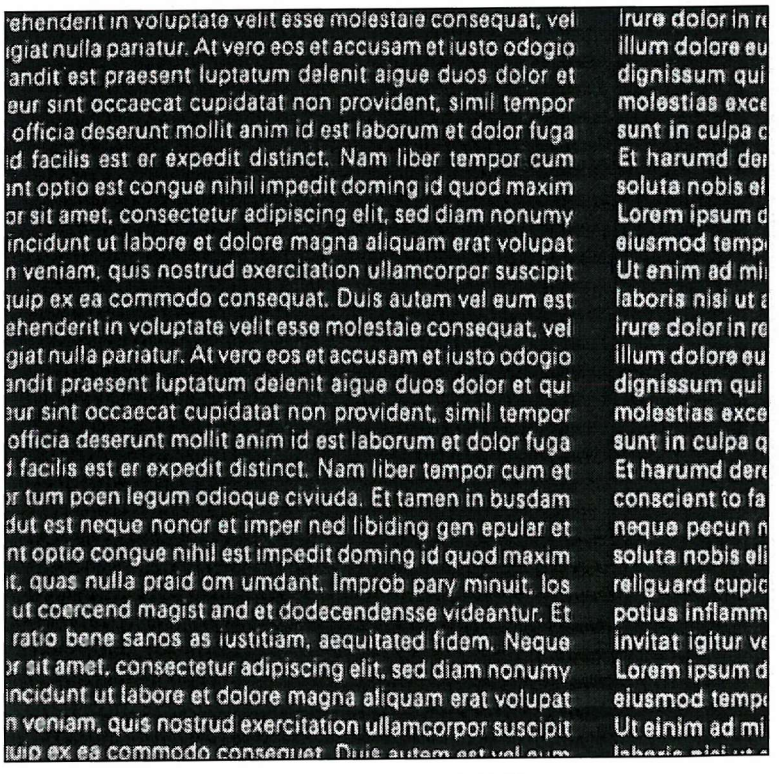

Figure 1.f. Frame 6 ("latin").

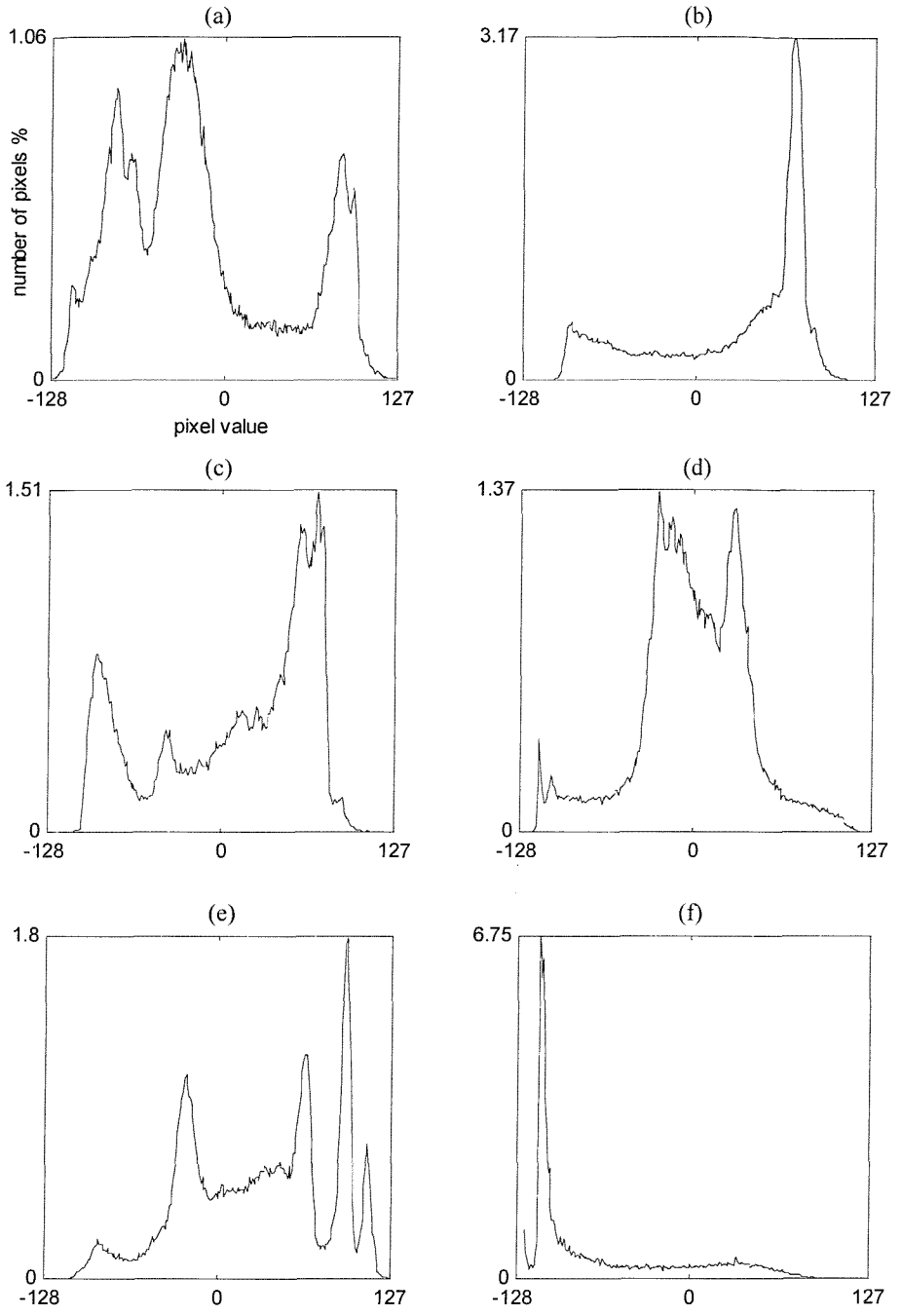

Figure 2. Frame set histograms: (a) frame 1, (b) frame 2, (c) frame 3, (d) frame 4 (e) frame 5, (f) frame 6.

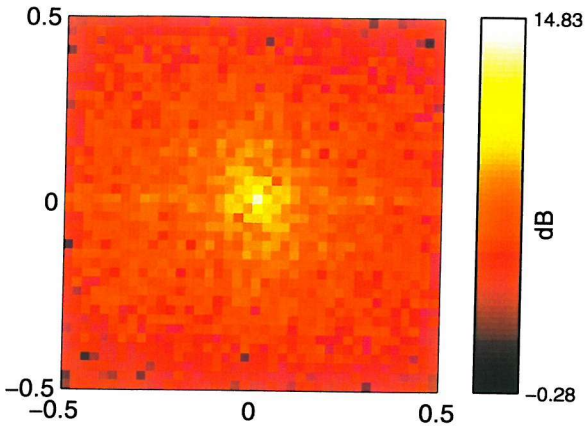

Figure 3.a. Frame 3 spectrum.

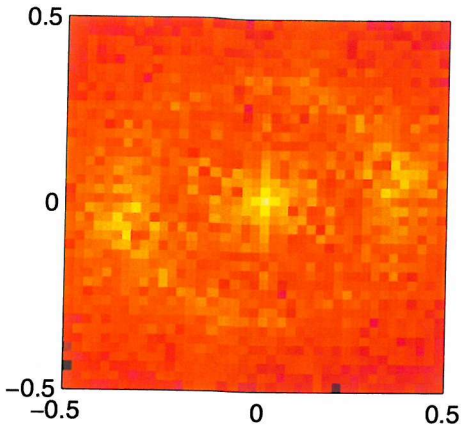

Figure 3.b. Frame 4 spectrum.

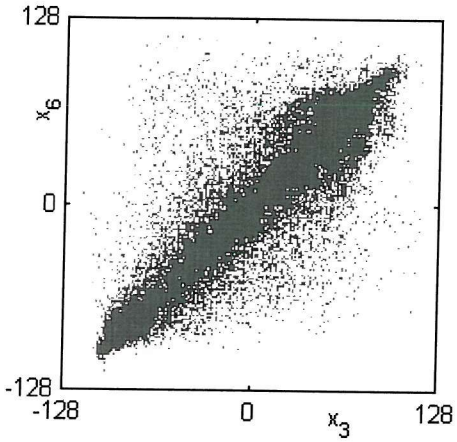

Figure 3.c. Frame 3 input distribution.

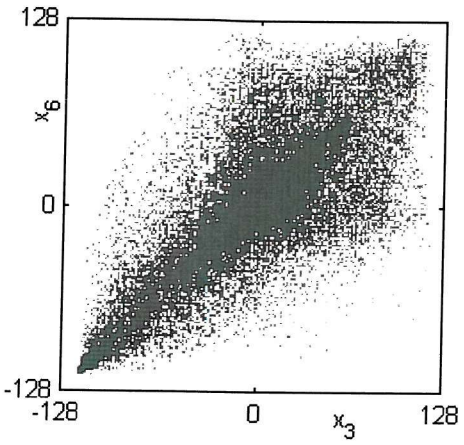

Figure 3.d. Frame 4 input distribution.

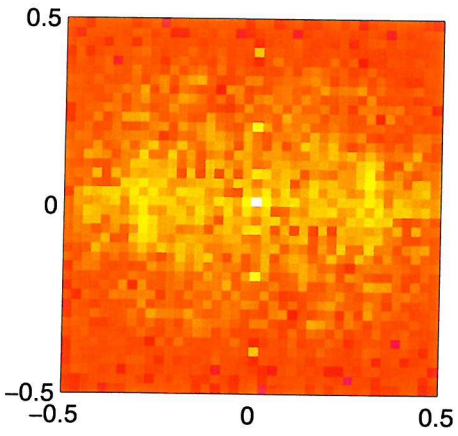

Figure 4.a. Frame 6 spectrum.

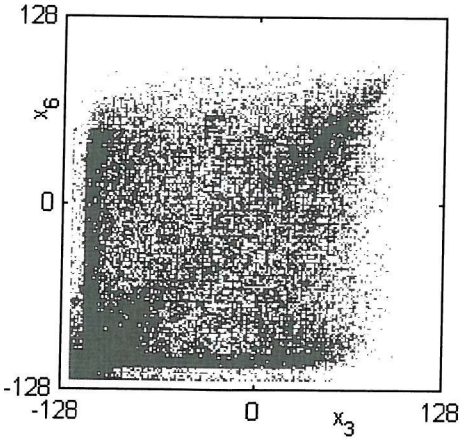

Figure 4.b. Frame 6 input distribution.

7.3. Bias and Variance

One of the key points to obtain a sufficiently general training is to make sure that the estimated mapping is sufficiently smooth, i.e. to ensure it does not over-fit the data. In order to predict unseen patterns, one should avoid too closely matching of the training patterns, since these may be not sufficiently representative of the full data. On the other hand, excessive smoothing of the mapping produces a poor result, albeit uniform over a wide range of inputs.

There are two key factors that govern this phenomenon. One is the order of the model, which dictates the size of the network. The other one is the training set, and its ability to represent the underlying data generator. Even if one had access to an exceptionally good set of training patterns, a poor interpolation maybe produced simply because the order of the model is insufficient to fit the data properly. Alternatively, a high-order model may over-fit the data. If these are noisy, or poorly represent general inputs, then the effect of the increased order is that the model fits the noise together with the data or, in the second case, that it specialises its mapping to an unrepresentative training set.

One way of measuring these effects is to decompose the error into bias and variance components (Geman *et al.*, 1992). In the case of a MSE training, the error can be written as (assuming an infinite training set):

$$C = \int \{ y(\mathbf{x}) - \langle t|\mathbf{x} \rangle \}^2 \, p(\mathbf{x}) d\mathbf{x} + \int \left\{ \left\langle t^2 | \mathbf{x} \right\rangle - \langle t|\mathbf{x} \rangle^2 \right\} p(\mathbf{x}) d\mathbf{x} \qquad (7.1)$$

where $p(\mathbf{x})$ is the density of the input set, $\langle t|\mathbf{x}\rangle \equiv \int t\, p(t|\mathbf{x}) d\mathbf{x}$ is the conditional average of the target set and $\left\langle t^2 | \mathbf{x} \right\rangle = \int t^2 p(t|\,\mathbf{x}) d\mathbf{x}$ is the conditional second-moment of the target.

As already discussed in chapter 2, the first term in (7.1) vanishes when the MMSE solution is reached:

$$y(\mathbf{x}) = \langle t | \mathbf{x} \rangle \tag{7.2}$$

The second term does not depend on the mapping $y(\mathbf{x})$. In fact, the second term represents the intrinsic noise in the data and sets the lower limit on the error that can be achieved (Bishop, 1995). In practical situations, the network is trained on a finite number N of patterns. Hence, the first term in (7.1) depends on the particular set chosen. If we consider an ensemble of T training sets, all made of N patterns, and all drawn from the same joint distribution $p(t, \mathbf{x})$, we can remove the dependency from the single training set by the ensemble average E_T over the ensemble T:

$$\int \mathsf{E}_T \left\{ [\, y(\mathbf{x}) - \langle t | \mathbf{x} \rangle \,]^2 \right\} p(\mathbf{x}) d\mathbf{x} \tag{7.3}$$

If the first term in (7.1) is zero for a particular training set, the ensemble (7.3) could still be greater than zero because (7.2) is not valid for the other sets. The quantity in (7.3) somehow represents the intrinsic ability of a particular model to characterise the system. In fact it averages the influence of the particular training set on the determination of the model. We can decompose (7.3) in bias and variance components:

$$\int \mathsf{E}_T \left\{ [\, y(\mathbf{x}) - \langle t | \mathbf{x} \rangle \,]^2 \right\} p(\mathbf{x}) d\mathbf{x} = \underbrace{\int \{ \mathsf{E}_T [\, y(\mathbf{x}) \,] - \langle t | \mathbf{x} \rangle \}^2 \, p(\mathbf{x}) d\mathbf{x}}_{(\text{bias})^2} + \tag{7.4}$$

$$\underbrace{\int \mathsf{E}_T \left\{ \{ y(\mathbf{x}) - \mathsf{E}_T [\, y(\mathbf{x}) \,] \}^2 \right\} p(\mathbf{x}) d\mathbf{x}}_{\text{variance}}$$

The details of this decomposition can be found in Bishop (1995). Equation (7.4) makes explicit why (7.3) could be greater than zero. Specifically, it makes explicit the influence of the model's order, and the influence of the training set.

Consider a situation when a low-order model, for instance a 1st-order polynomial, is trained to interpolate a 3rd-order system (see figure 5.a). The T training sets are all supposed to be appropriate, e.g. with a low level of noise. In this case, the variance will be very small, since all the sets generate approximately the same mapping $y(\mathbf{x}) \cong \mathsf{E}_T[y(\mathbf{x})]$. Conversely, the bias will be high since the insufficient order of the model will generate mappings that are on average very different from the "true" mapping $\langle t|\mathbf{x}\rangle$.

The opposite happens when the model order is higher than the system (e.g. 10th-order model), and the set of training patterns is noisy (see figure 5.b). The effect of noise will be cancelled out by the ensemble average, and the bias will be negligible. On the other hand, the high order will make the model over-fit the data. Each individual mapping $y(\mathbf{x})$ will be very different from the average $\mathsf{E}_T[y(\mathbf{x})]$, and the variance will be high.

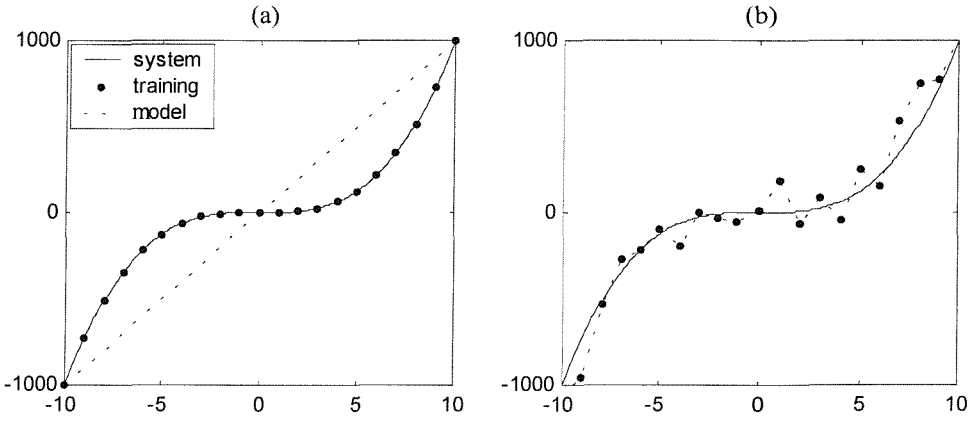

Figure 5. Polynomial fitting. (a) linear approximation to a cubic (b) 10th-order polynomial approximation to a cubic.

One might assume that the two quantities trade-off against each other, although it is not necessarily so. For instance, a model might have high bias and variance, if its order is insufficient and it is trained on noisy data. On the other hand, it is possible for

both quantities to be small if the model is accurately chosen to match the system's complexity, and the training sets are properly chosen.

In the case of RBFN, one expects the models to have higher variance and lower bias when the number of nodes is high, and the contrary to be true when the number of nodes is small. In chapter 6 it was shown that a RBFN with a large number of nodes has sufficient complexity to produce accurate mappings of the training set considered. We will shortly see that the price paid for this accuracy is that the model specialises on the training set, producing poor results when attempting to interpolate a new set of inputs.

Given the previous considerations, one way to reduce variance keeping the bias low is to join training sets, i.e. creating an ensemble of sets that are the union of two or more frames. In this case, the sets are more uniform, therefore reducing the variance. As already stated, this approach is limited by computational issues, and does not produce "built-in" generalisation.

7.4. Regularisation

It has been shown how a network that over-fits the training set may produce insufficiently general results. Therefore we should try to limit the tendency of the training to over-fit the data.

One way of doing so is by means of regularisation (also see section 3.9). Regularisation has been introduced in chapter 3, in relation to RBFN. Regularisation is a way to control the smoothness of an interpolating mapping, and as such is clearly a generalisation technique since it tends to cancel out high-curvature over-fitting. The technique consists of adding a penality term to the cost function, so that the tendency of the network to minimise the error at the cost of smoothness is counter-balanced by a factor that increases as smoothness decreases:

$$C = C_{ERR}\left[t(\mathbf{x}), y(\mathbf{x})\right] + \nu \cdot \Omega \qquad (7.5)$$

Where $C_{ERR}[t(\mathbf{x}), y(\mathbf{x})]$ is the unregularised cost function (e.g. the MSE), that typically depends on how close the mapping $y(\mathbf{x})$ is to the particular training pattern $t(\mathbf{x})$. Conversely the regularisation term, Ω, is a function of only the network's mapping. The contribution of the regularisation term is controlled by the regularisation factor ν. In general, the unregularised cost function decreases as the training progresses, while the regularisation term increases as the mapping approaches the training set. A minimum of C can be found that is a compromise between the accuracy of the mapping and its ability to generalise (figure 6). Examples of regularised cost functions have been given in section 3.9. In this chapter a simple regulariser is applied: the weight decay training technique.

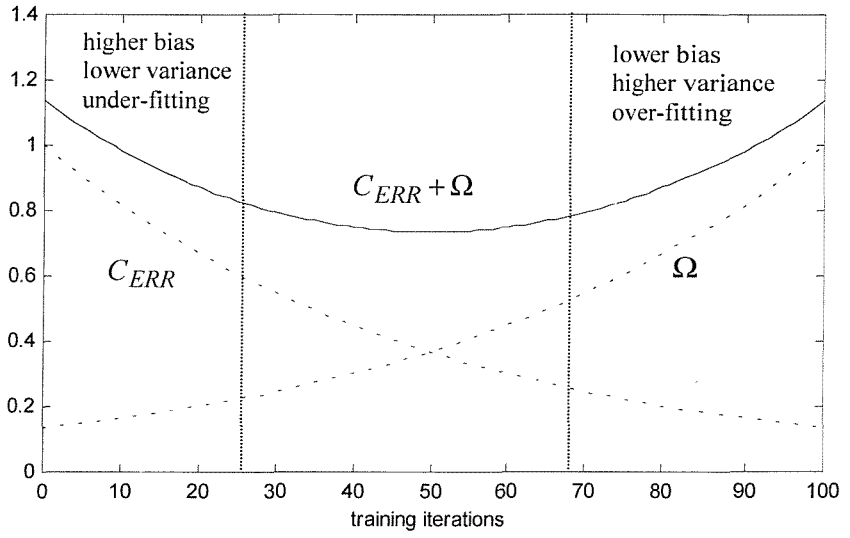

Figure 6. Illustrative example of regularised training for a general iterative optimisation.

7.4.1. Weight decay training

One of the simplest forms of regulariser is weight decay, which uses a penalty term consisting of the sum of the squared values of the linear weights:

$$\Omega = \frac{1}{2}\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi} = \frac{1}{2}\sum_{i=1}^{M}\phi_i^2 \tag{7.6}$$

This constitutes a regulariser in the sense of Arsenin and Tikhonov (Tikhonov, Arsenin, 1977). The subsequent regularised minimisation function is referred as *ridge regression* (Bishop, 1995; Smola *et al.*, 1996). The empirical justification for this form of regulariser is that over-fitted mappings with large curvatures usually require large values of the weights. By using in (7.5) the regulariser given by (7.6), the training algorithm is encouraged to produce small weights and consequently, smoother mappings. It has been shown that a regulariser of this form can lead to significant improvements in the generalisation ability of a mapping (Hinton, 1987).

If $C_{ERR}(t,\mathbf{x})$ is the sum-of-squares error function, insights into the effect of weight decay can be drawn. In the absence of regularisation, the error function can be written as a quadratic form:

$$C_{ERR} = \frac{1}{2}\mathbf{\Phi}^T\mathbf{R}\mathbf{\Phi} - \mathbf{P}^T\mathbf{\Phi} + const \tag{7.7}$$

The solution that minimises (7.7) is the familiar one:

$$\mathbf{R}\mathbf{\Phi} - \mathbf{P} = 0 \Rightarrow \mathbf{\Phi} = \mathbf{R}^{-1}\mathbf{P} \tag{7.8}$$

In presence of the weight decay regularisation term, it is easy to show that the solution is given by:

$$(\mathbf{R} + \nu \cdot \mathbf{I})\mathbf{\Phi} - \mathbf{P} = 0 \Rightarrow \mathbf{\Phi} = (\mathbf{R} + \nu \cdot \mathbf{I})^{-1}\mathbf{P} \tag{7.9}$$

Hence, the solution moves away from the minimum of the error function, to a sub-optimal solution that trades accuracy with generalisation. Equation (7.9) also provides a direct implementation of the algorithm, by adding the factor ν to the diagonal elements of $\mathbf{R}$.

7.4.2. Orthogonal weight decay training

Having described the weight decay technique, we seek to apply it to the OLS algorithm described in chapter 4. At each step k of the orthogonalisation, the selected regressors are arranged into an orthogonal set $\mathbf{W}^{(k)}$, and $\mathbf{R}$ is diagonal with non-null elements $r_{ii} = \mathbf{w}_i^T \mathbf{w}_i$. Consequently, the error reduction ratio given by substituting (4.4) in (4.5) will be modified by weight decay to become:

$$err_i^{(k)} = \frac{\left(\mathbf{T}^T \mathbf{w}_i^{(k)}\right)^2 \left(\mathbf{w}_i^{(k)T} \mathbf{w}_i^{(k)}\right)}{\left(\mathbf{w}_i^{(k)T} \mathbf{w}_i^{(k)} + v\right)^2 \left(\mathbf{T}^T \mathbf{T}\right)} \tag{7.10}$$

7.4.3. Support Vector Machines

An alternative viewpoint to the problem of generalisation is given by the theory of Support Vector machines (SV) (Smola *et al.* 1996; Vapnik, *et al.* 1997; Drucker *et al.* 1997, Smola, Scholkopf, 1998). The SV theory is a non-linear generalisation of the *Generalised Portrait* algorithm (Vapnik, Lerner, 1963; Vapnik, Chervonenkis, 1964). In its most general form, the SV regression algorithm tries to determine the mapping $y(\mathbf{x}) = \mathbf{\Phi}^T \mathbf{h}(\mathbf{x})$, $\mathbf{h} \in R^M$ with maximum *flatness*, constrained to some condition on the mapping's error, $C(e) = C(t - y)$. The flatness is defined as the squared sum of the weights as in (7.6) (Smola *et al.* 1996). Therefore, the flattest mapping is the line with minimum slope in R^M that satisfies the constraining condition.

Different to conventional regularisation techniques, in SV theory the function to minimise is the squared sum of weights (7.6), which therefore is called the *primal objective* function, while $C(e)$ represents a condition (constraint) for the minimisation problem. Hence, the SV problem is a constrained minimisation of (7.6), where the constraint can be, for instance, the maximum permitted value of the error function.

It can be shown that there are several connections between the SV theory and regularisation. In particular, if $C(e)$ is the standard MSE error, the two approaches

lead to the same algorithm (Smola, Scholkopf, 1998). For this reason, we will not investigate the application of SV theory to the solution of our problem, but rather apply the weight-decay technique that has the minimisation of the MSE as its main goal.

7.5. Individual results

In this section we will present the performance of the system trained on individual images and then applied to the wider data set. In other words, we will examine how each of the frames 1 to 6 can be considered as being representative of the other frames. The systems investigated will be the linear filter, the 3rd-order Volterra series, and the HGRBFN using the 8_1 sampling lattice.

These experiments will show how inappropriate training leads to poorly generalised mappings and, more importantly, how non-linear performance is affected. As the order of the system increases, an unrepresentative training set produces less general results.

7.5.1. Linear filters

Table 1.a shows the MSE of 6 linear filters. Each filter is trained using one image and its performance is measured for all 6 images. The results are arranged in a matrix whose rows represent the individual filters (i.e. the training sets) and the columns the resulting normalised MSE on the frames. Each column j is normalised with respect to its j-th element. In other words, the ability of the filter i to de-interlace the frame j (hereinafter referred a "cross-result") is compared to the result obtained with the filter j, which will necessarily produce the most accurate result ("self-result"), since it has been trained on the j-th frame.

One can see how the performance is similar for all the filters. From table 1.b, one can see that frame 2 produces the best overall performance, although the differences are not significant. Note that in table 1.b, the self-result has not been included in the calculation.

		input image					
		1	2	3	4	5	6
Filter	1	1	1.0591	1.2518	1.0964	1.1577	1.1541
	2	1.0364	1	1.0438	1.0304	1.0169	1.0228
	3	1.0865	1.0265	1	1.0889	1.0080	1.0188
	4	1.0891	1.0552	1.2177	1	1.1359	1.0600
	5	1.0625	1.0163	1.0118	1.0729	1	1.0243
	6	1.0902	1.0250	1.0333	1.0522	1.0260	1

Table 1.a. Linear results.

	Filter					
	1	2	3	4	5	6
mean	1.1438	1.0301	1.0457	1.1116	1.0376	1.0453
stdev	0.0731	0.0107	0.0389	0.0674	0.0281	0.0274

Table 1.b. Linear results, average and standard deviation.

7.5.2. Volterra series

Tables 2 show the corresponding results for a 3rd-order Volterra model. In table 2.a the self-results are normalised with respect to the corresponding linear self-result, whilst in table 2.b and c the cross-results are normalised with respect to the corresponding Volterra self-result. The situation observed in the last section changes dramatically when a 3rd-order Volterra series is examined. The results noticeably differ and in some cases there is a dramatic effect on performance with a complete lack of generalisation.

As anticipated in previous sections, the most peculiar behaviour is shown by filter 6, which has been trained on the text image. This series exhibits the largest performance increase compared to the linear filter (table 2.a), but it also exhibits a very poor ability to generalise (table 2.b). It is evident that the attempt of the series to match such a peculiar pattern produces an over-specialised (i.e. over-fitted) map, that is unable to estimate patterns different to those included in the training set. Unsurprisingly, the other series poorly estimate frame 6. Patterns like those encountered in frame 6 are generally rarely encountered in the other frames. Therefore, their density $p(\mathbf{x})$ is small, and they scarcely influence the minimisation of the error.

		input image					
		1	2	3	4	5	6
	ratio	0.7745	0.8463	0.7726	0.7865	0.8658	0.4789

Table 2.a. Volterra/Linear ratio (self-results).

		input image					
		1	2	3	4	5	6
series	1	1	1.1520	1.3615	1.5163	1.3593	2.4340
	2	1.2002	1	1.1257	1.3046	1.2615	2.0619
	3	1.7131	1.2051	1	1.7228	1.2184	3.9398
	4	2.4079	2.0198	3.1139	1	2.7101	3.7983
	5	2.3924	1.3673	1.2587	2.5180	1	11.2321
	6	13.4649	18.8674	40.7060	5.9126	121.3406	1

Table 2.b. Volterra results (normalised to self-results).

	filter					
	1	2	3	4	5	6
mean	1.5646	1.3908	1.9598	2.8100	3.7537	40.0583
stdev	0.5029	0.3812	1.1354	0.6829	4.2198	47.2478

Table 2.c. Volterra results, average and standard deviation.

A similar behaviour is found, although not in such dramatic way, for series 4. The presence of high frequency patterns affects the general performance of the mapping. An opposite behaviour can be conjectured for series 5 ("face"). The prevalence of uniform areas in the training set produces a mapping that lacks precision on the images with a preponderance of high contrast areas (frame 1, 4 and 6). Series 1, 2 and 3 show a reasonably stable behaviour over the frame set, albeit frame 6 remains difficult to interpolate. The average results are shown in table 2.c.

These simulations clearly highlight the degradation in generalisation that using a Volterra series introduces, as compared to a linear de-interlacer.

7.5.3. Hybrid GRBFN

A corresponding behaviour is found in the application of RBFN techniques. Tables 3 show the results using a hybrid network with OLS training. As the model order (i.e. the number of centres) increases, the performance of the mapping on the training pattern improves and produces a corresponding reduction in the ability to generalise. Again, networks trained on frame 6 show the most peculiar behaviour.

M=10		input image					
		1	2	3	4	5	6
network	1	1	1.0554	1.2550	1.1444	1.2240	1.3167
	2	1.0627	1	1.0539	1.0944	1.2467	1.1967
	3	1.1068	1.0247	1	1.1489	1.1520	1.1527
	4	1.1157	1.1458	1.4829	1	1.2153	1.1501
	5	1.0770	1.0187	1.0223	1.0959	1	1.1612
	6	2.5660	5.1117	13.5979	2.9994	29.4186	1

M=25		input image					
		1	2	3	4	5	6
network	1	1	1.1293	1.5442	1.4169	1.5599	1.6066
	2	1.1341	1	1.0724	1.1649	1.2214	1.5905
	3	1.1822	1.0461	1	1.2436	1.0846	1.4409
	4	1.2709	1.3196	2.0299	1	4.1006	1.4587
	5	1.1782	1.0315	1.0343	1.2087	1	1.4867
	6	3.7294	4.1191	9.3875	2.7428	56.0766	1

M=50		input image					
		1	2	3	4	5	6
network	1	1	1.1935	1.7192	1.5693	1.6573	1.9771
	2	1.1866	1	1.1414	1.2603	1.2787	1.9364
	3	1.2593	1.0929	1	1.4418	1.2700	2.0237
	4	1.5774	1.5675	2.7314	1	14.3283	2.0534
	5	1.2715	1.1048	1.1807	1.3634	1	1.9777
	6	3.6851	3.4543	8.4246	3.4752	27.8970	1

M=100		input image					
		1	2	3	4	5	6
network	1	1	1.2624	1.6620	1.5920	1.7430	2.6317
	2	1.2823	1	1.2071	1.3844	2.4735	2.2670
	3	1.3413	1.1346	1	1.5306	1.2584	2.5933
	4	1.7150	1.7249	2.9347	1	7.6300	2.4784
	5	1.3658	1.2012	1.2947	1.5385	1	2.3311
	6	5.2940	5.7335	16.0688	4.7179	42.8350	1

Table 3.a. OLS cross-results, normalised to self-results. 10 (top), 25, 50 and 100 (bottom) centres.

		network					
		1	2	3	4	5	6
centres	10	1.1991	1.1309	1.1170	1.2220	1.0750	10.7387
	25	1.4514	1.2367	1.1995	2.0359	1.1879	15.2111
	50	1.6233	1.3607	1.4175	4.4516	1.3796	9.3872
	100	1.7782	1.7229	1.5716	3.2966	1.5463	14.9298

Table 3.b. OLS mean results, normalised to self-results.

	input image					
	1	2	3	4	5	6
10	0.9804	0.9903	0.9924	0.949	0.9826	0.8901
25	0.8922	0.9662	0.9625	0.8725	0.9579	0.7144
50	0.8183	0.905	0.8422	0.8096	0.9299	0.5529
100	0.7388	0.8422	0.7759	0.7457	0.871	0.4472

Table 3.c. RBFN/linear ratio sequence.

	input image					
	1	2	3	4	5	6
10	1.266	1.1702	1.2844	1.2066	1.135	1.8585
25	1.152	1.1416	1.2458	1.1093	1.1064	1.4917
50	1.0566	1.0694	1.0901	1.0294	1.0741	1.1544
100	0.954	0.9951	1.0042	0.9482	1.006	0.9338

Table 3.d. RBFN/Volterra ratio sequence.

Attempting to produce more general results by reducing the network's size is obviously limited by the general increase of the error. However, these results confirm the observations on the increasing specialisation, since as the number of centres decreases, the network is forced to map the general pattern of the data common to all frames considered, rather than the specific, particular patterns in the training set. This point will be re-addressed when the bias and variance for the orthogonal sequence are evaluated (section 7.5.5).

7.5.4. Linear and non-linear parameters

The Volterra series uses a fixed non-linear structure and produces poor general mapping but the best self-results. This is because the non-linear layer is, in principle, able to estimate any pattern, and therefore is highly general. It is therefore the weights that, trained to match a specific set, are responsible for the lack of generalisation. From this point of view, one might assume that the generalisation problem is a linear one. Techniques that achieve more general results by linear means (e.g. the weight-decay technique illustrated in section 7.4.1) might be applied successfully to improve performance.

In RBFN, the picture is different. The non-linear parameters (i.e. centres and widths) are related to the training set since the distribution of centres approximates the distribution of inputs. This is especially true when the initial centres are chosen from

the training set. Furthermore the application of OLS may reduce the generalisation of the non-linear layer. Nodes in an initially large network are chosen only according to their ability to describe that particular training set. Therefore, nodes that might prove useful to generalise the result may yet be discarded by OLS.

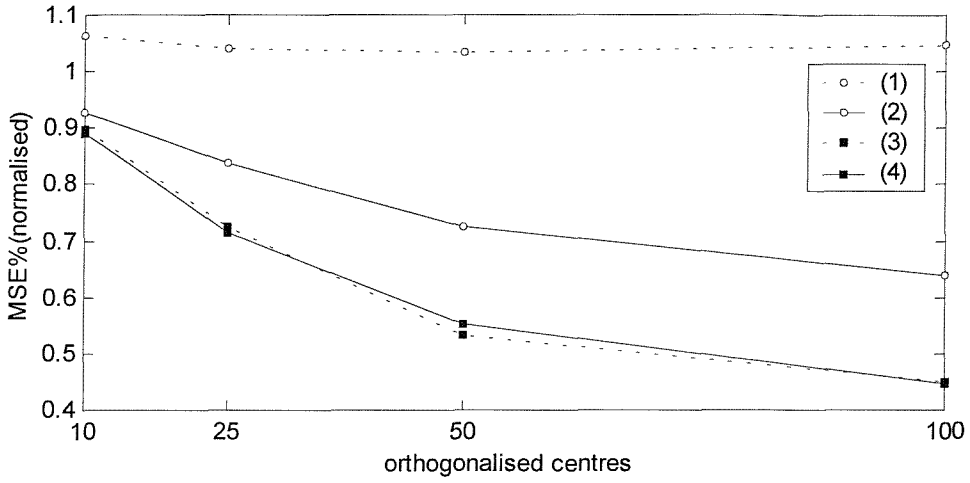

Figure 7. Network trained on frame 2 and tested on frame 6. Normalised with respect to the linear self-result. (1) orthogonalisation, weights and centres from frame 2. (2) weights calculated on frame 6. (3) orthogonalisation and weight calculation based on frame 6. (4) frame 6 self-result.

It is then reasonable to explore how the generalisation ability of the network is determined by the linear and non-linear layers, and how OLS influences the results. Figure 7 shows the results of the following experiment. A number of centres (1000) is randomly selected from frame 2, from which three networks are created. The first is trained on frame 2, using OLS; the second is initially orthogonalised on frame 2, but the weights are re-calculated from frame 6. Hence the centres are selected from and optimised for frame 2, but the weights are optimised for frame 6. Finally, the last network is orthogonalised and trained on frame 6, although its initial centres are still drawn from frame 2. The performance of the three networks is evaluated on frame 6, and compared with frame 6 self-result. The MSE is normalised with respect to the MSE of the linear self-result.

These results show how the three phases of the training (selection of centres, orthogonalisation, and weight's calculation) determine the generalisation ability of the model. The first network, completely unrelated to frame 6, produces a result that is worse than that obtained with the linear filter. The second network is the most

interesting case, since although the centres are drawn from frame 2, and orthogonalised on it, the recalculation of weights based on frame 6 produces a reasonable result. (This is an understandable result, since the exceptional self-result of network 6 is largely due to over-fitting). Hence centres whose position is optimised for frame 2 still produce a good result if the weights are properly "re-tuned". Finally, the full OLS training on frame 6 of centres initially drawn from frame 2 produces a result that is virtually identical to that produced by network 6. This suggests that the source of the initial population of 1000 centres is unimportant.

Therefore it is reasonable to assume that in the RBFN case, the problem of generalisation can be addressed by investigating more general training procedures for the linear layer. In other words, the experiment shows that the initial choice of centres is not as critical, in terms of generalisation, as the determination of weights.

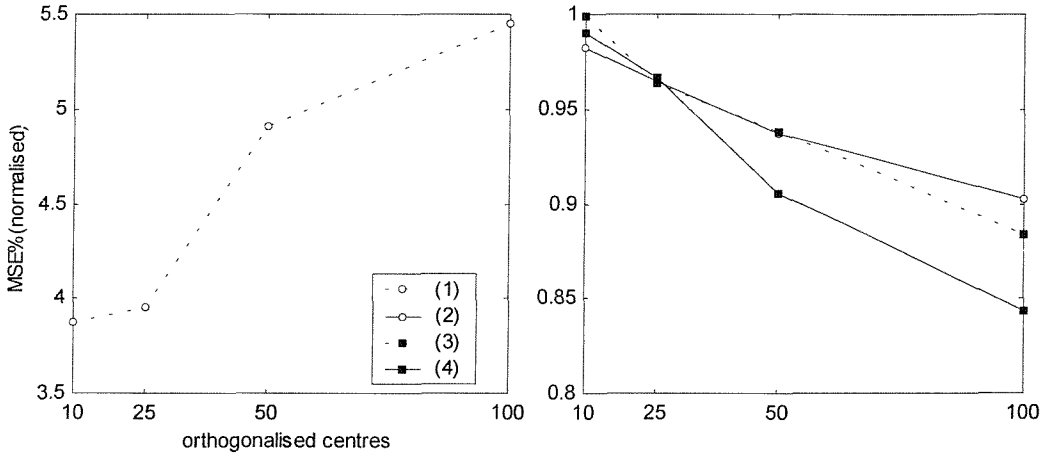

Figure 8. Network trained on frame 6 and tested on frame 2: (1) orthogonalisation, weights and centres from frame 6. (2) weights calculated on frame 2. (3) orthogonalisation and weight calculation on frame 2. (4) frame 2 self-result.

One should be careful however that the initial centres are chosen from a sufficiently general set. The same experiment is repeated with the role of frame 2 and frame 6 reversed. The results are shown in figure 8, that is divided in two separate plots because of the different scales needed to represent the curves. This time, the orthogonalisation on frame 2 of centres selected from frame 6 produces a result that is very different from the frame 2 self-result. Therefore, we must assume that the random selection of centres from frame 6 does not produce a generalised set, and in

this case the generalisation problem is truly non-linear, a situation to be avoided where possible.

7.5.5. Bias and variance

This section investigates how the bias and variance change as the order of the model (i.e. number of centres) increases. Specifically, we seek to calculate these two quantities for a model attempting to interpolate frame 2, i.e. compute the two terms in (7.4). The main problem is the determination of the "true" mapping $\langle t | \mathbf{x} \rangle$, since this is the overall goal of the thesis.

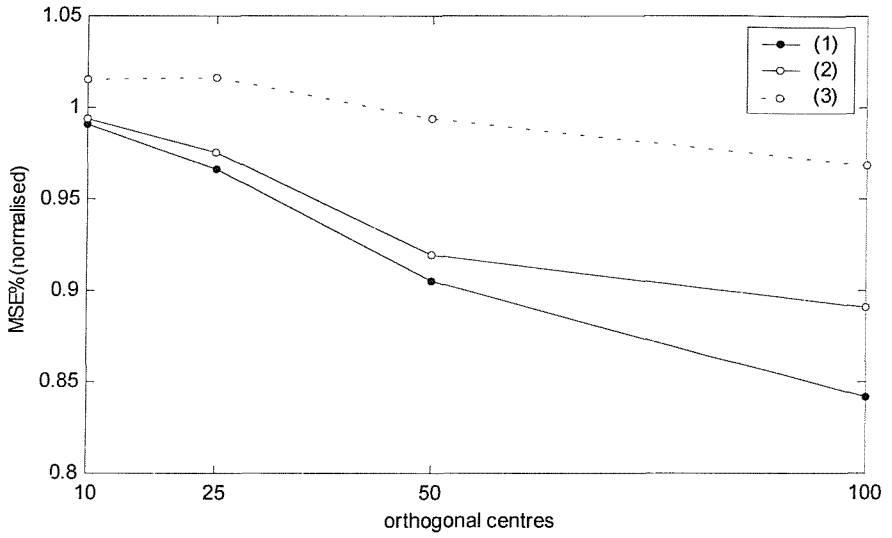

Figure 9. Training on the full frame set. (1) Self-result of frame 2. (2) Centres selected from frame 2, and orthogonalised on the whole frame set (frame 6 excluded). (3) Average of the cross-results on frame 2. (frame 2 and 6 excluded).

One way to achieve an estimate of the true mapping is by training 100 centres, orthogonalised from frame 2, on a training set consisting of the set of 5 frames (frame 6 being excluded as non-representative of a general image set). In this way the mapping will not over-fit frame 2, since it has been trained on a larger set. One might question whether 100 centres are sufficient to guarantee a low bias (i.e. if the model order does not over-smooth the set and hence frame 2), but the objective is to validate the discussion in section 7.3, so that shortcomings in our absolute measures of bias and variance are not critical. Figure 9 plots the full-training MSE computed on frame 2 and normalised with respect to the linear self-result for various network sizes. From

figure 9 one can see that the training on the larger frame set produces sub-optimal results compared to frame 2 self-result[†].

The individual responses $y(\mathbf{x})$ are calculated by firstly training the centres on frames 1 to 5 (hence creating individual mappings optimised on these frames), and then using frame 2 as input. The results are averaged to estimate $\mathsf{E}_T[y(\mathbf{x})]$. Frame 6 is excluded from all the calculations, since it is not representative of frame 2 and it would only generate unwanted outliers.

The density of $\mathbf{x}$ is considered uniform over L, the number of pixels of the training set, $p(\mathbf{x}) = L^{-1}$, although this is a rather simplistic assumption. The results are shown in figure 10. The plots are normalised with respect to their maximum value. It is evident that the plots conform to those anticipated in light of the discussion in section 7.3, despite of the simplistic method used. Increasing the number of centres reduces the bias since the model's order is increased, but conversely this makes the model more sensitive to the specific training pattern and increases the variance.

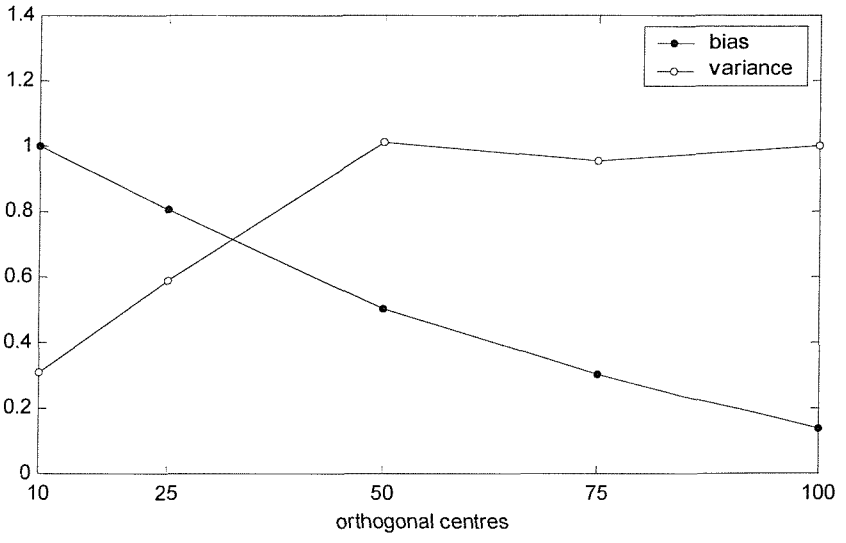

Figure 10. Bias and variance for the orthogonal model aimed at interpolating frame 2.

[†] One may question why we do not use this extended training model in the first place. The point is that we want to devise procedures that give intrinsic generalisation for limited training sets.

7.6. Weight-decay training of Volterra series and RBFN

This section considers the application of weight decay techniques described in section 7.4.1, to Volterra series and RBFN. In figure 11 the average output error is shown (cross-results and self-result) for each series, vs. the regularisation factor. The plots also show the minimum and maximum values of the cross-result. The performances of series 1 to 5 on frame 6 have been purposely omitted from the results, since they generate unwanted outliers. All the sequences are normalised with respect to their non-regularised MSE.

From figure 11 it is clear that frames 1, 2 and 3 produce the most general training sets. The regularisation of their respective series produces little improvements to their general performance. On the other hand, regularisation improves the performance of non-general series, like 4, 5 and 6. However, the improvement is insufficient to consider the corresponding series of practical use.

A similar result is obtained when regularisation is applied to the orthogonal training of the RBFN. One should remember that weight decay is a linear technique, hence the centres are not directly optimised to produce smoother mappings. However, since regularisation is applied to the OLS procedure through equation (7.10), in some way we also choose centres that, once trained, yield smaller weights.

Figure 12 and 13 show the results obtained using 100 and 25 centres. Note how network 6 yields a relatively better result than the corresponding Volterra series. One should be careful to consider that these results depend on the initial random selection of centres (chapter 6), therefore it is inappropriate to draw any general conclusions from this result.

Note also how the reduction of the number of centres improves the generalisation performance of all the networks. We have already seen that this approach is limited by the fact that the overall performance decreases when M is reduced. It has been demonstrated that this is due to the decreased order of the model, and the consequent decrease in variance.

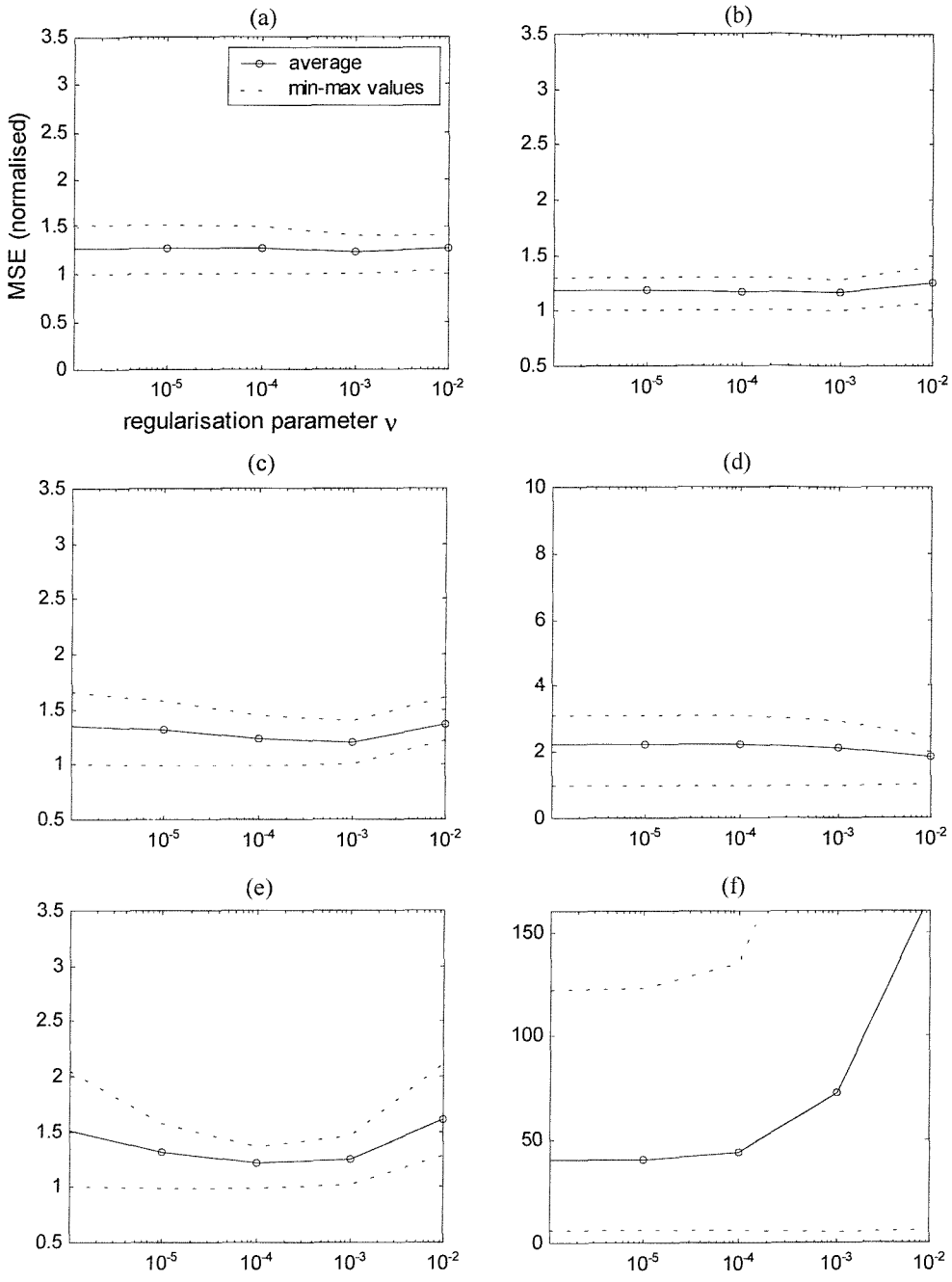

Figure 11. Regularisation of Volterra series. Mean, minimum and maximum values over frames 1 to 5. (a) Frame 1, (b) frame 2, (c) frame 3, (d) frame 4, (e) frame 5, (f) frame 6.

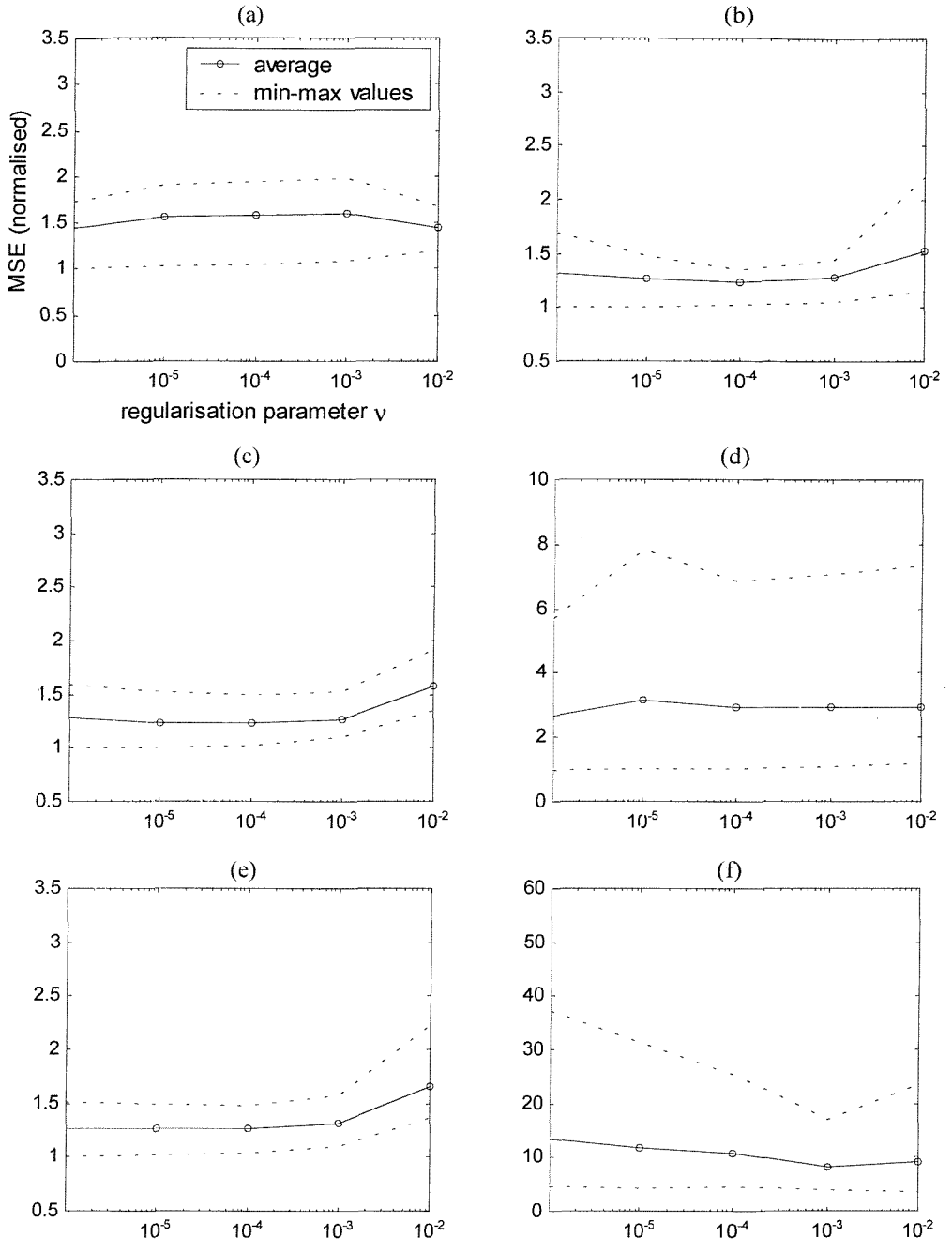

Figure 12. Regularisation of RBFN series, 100 OLS centres. Mean, minimum and maximum values over frames 1 to 5. (a) Frame 1, (b) frame 2, (c) frame 3, (d) frame 4, (e) frame 5, (f) frame 6.

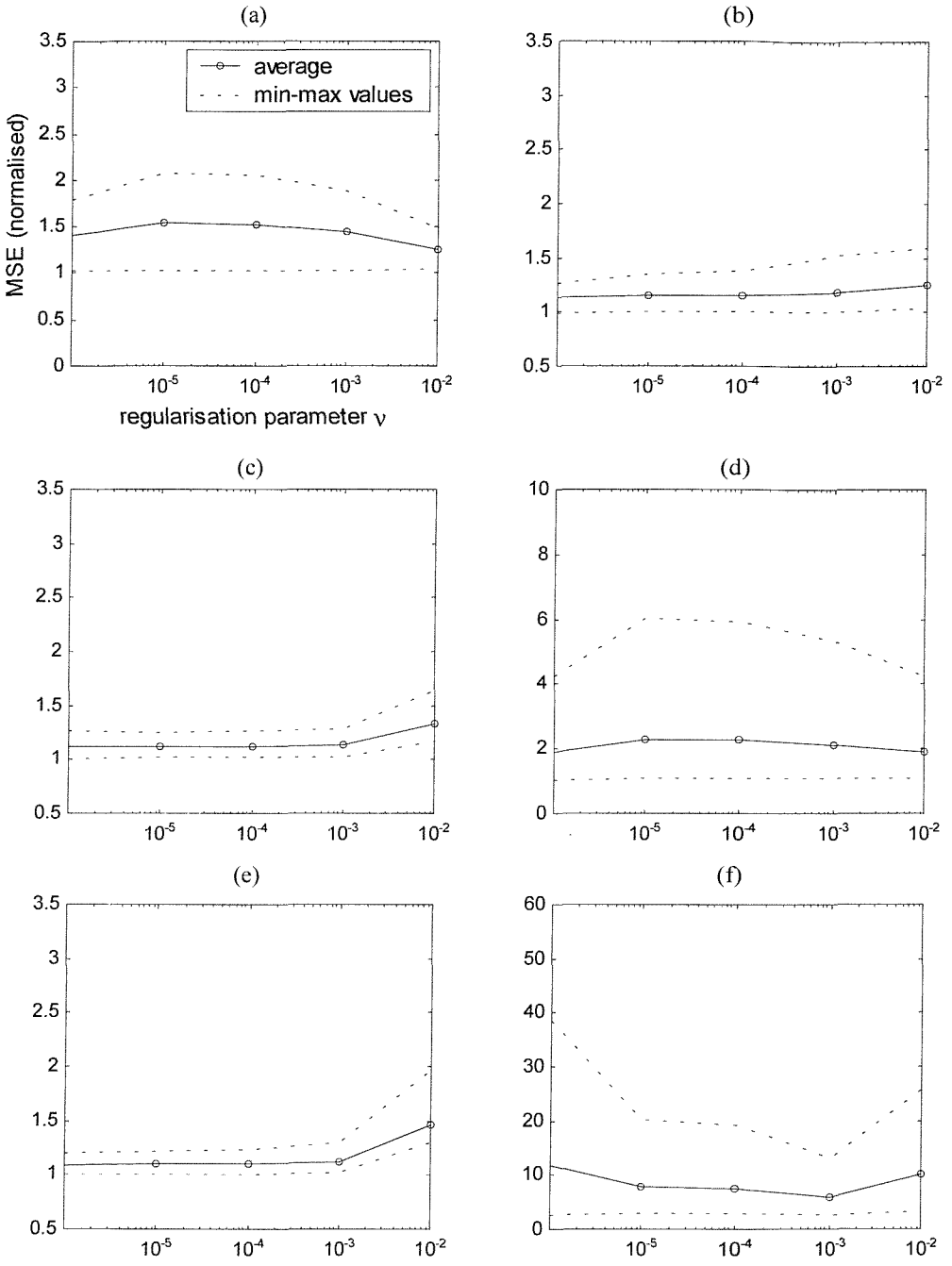

Figure 13. Regularisation of RBFN series, 25 OLS centres. Mean, minimum and maximum values over frames 1 to 5. (a) Frame 1, (b) frame 2, (c) frame 3, (d) frame 4, (e) frame 5, (f) frame 6.

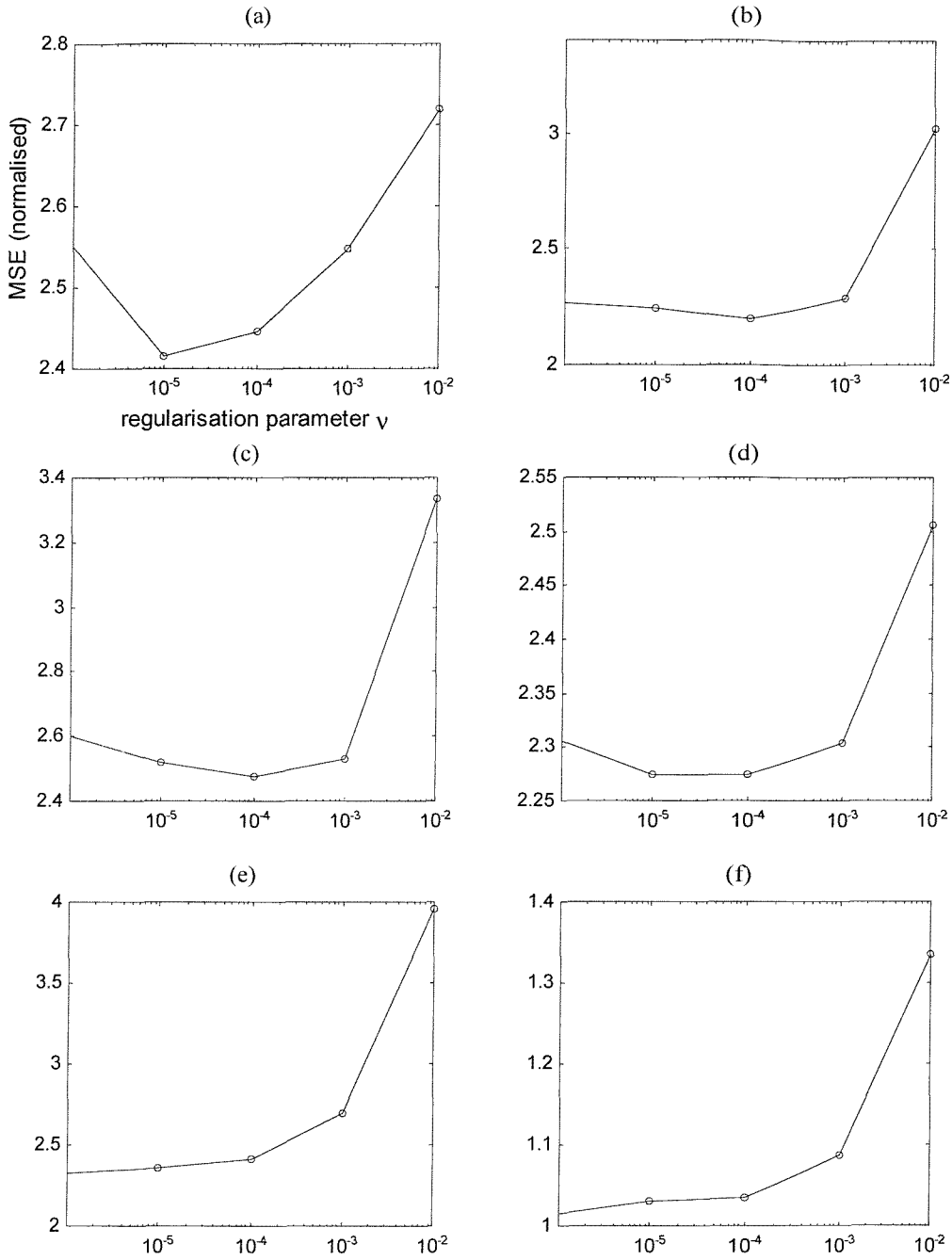

Figure 14. Regularisation of RBFN series, 100 OLS centres with frame 6 as input. (a) Frame 1, (b) frame 2, (c) frame 3, (d) frame 4, (e) frame 5, (f) frame 6.

7.7. Extended training set

The preceeding sections demonstrated that it is difficult to improve the performance of the interpolator with the training techniques adopted. Training based on frame 2 for instance is sufficiently general to produce reasonable results over frames with similar characteristics, e.g. frame 1 and 3. Nevertheless, it proves ineffective on images like frame 6. Figure 14 shows the regularisation sequence for RBFN networks 1 to 6, when frame 6 (excluded from figures 12 and 13) is the input. The values are normalised with respect to the network 6 self-result. It is clear that, although regularisation may produce some improvement, good performances achieved by training on that frame is never repeated, and a specialised network is required to obtain good results for this frame.

In fact, reaching such result requires a non-smooth mapping, which makes weight-decay counter-productive. It is unclear whether training on a single frame is sufficient, and whether a single network can cope with such differing inputs. In this section, we will address the first of these questions.

The goal of this section is to produce a training that minimises the MSE on a combination of patterns from frame 2 and frame 6. However, we will use the non-linear structure of network 2, so the initial set of 1000 centres will be drawn from frame 2. It has been shown that this selection is not critical to the performance on frame 6. A weighted sum of two correlation matrices **R** and two cross-correlation vectors **P** is formed. The two correlation structures are computed from the non-linear layer, when the inputs are respectively frames 2 and 6.

$$\begin{aligned}\mathbf{R} &= (1-\nu)\cdot \mathbf{H}^T(\mathbf{x})\mathbf{H}(\mathbf{x})\Big|_{\mathbf{x}\in frame\,2} + \nu\cdot \mathbf{H}^T(\mathbf{x})\mathbf{H}(\mathbf{x})\Big|_{\mathbf{x}\in frame\,6} \\ \mathbf{P} &= (1-\nu)\cdot \mathbf{H}^T(\mathbf{x})t(\mathbf{x})\Big|_{\mathbf{x},t\in frame\,2} + \nu\cdot \mathbf{H}^T(\mathbf{x})t(\mathbf{x})\Big|_{\mathbf{x},t\in frame\,6}\end{aligned} \tag{7.11}$$

with $\nu \in [0,1]$. Finally, the OLS algorithm is applied in its matrix form (chapter 4, section 3) to orthogonalise the matrices **R** and **P**. Note that (7.11) has the formal structure of a regularisation. One can assume that the solution of (7.11) forces a network designed to minimise the error on frame 2 to yield a reasonable result on

frame 6 (or vice-versa, given the symmetric form of the regularisation). In fact, $\mathbf{\Phi} = \mathbf{R}^{-1}\mathbf{P}$ with $\mathbf{R}$ and $\mathbf{P}$ as in (7.11) is the MMSE solution of the error function:

$$(1-\nu)\cdot\sum_{n=0}^{L-1} e_n^2\Big|_{frame2} + \nu\cdot\sum_{n=0}^{L-1} e_n^2\Big|_{frame6} = \tag{7.12}$$
$$(1-\nu)\cdot\sum_{n=0}^{L-1}[y(\mathbf{x}_n)-t_n]^2\Big|_{\mathbf{x}_n,t_n\in frame2} + \nu\cdot\sum_{n=0}^{L-1}[y(\mathbf{x}_n)-t_n]^2\Big|_{\mathbf{x}_n,t_n\in frame6}$$

(It has been assumed for simplicity that the two training sets have the same length L). This formal analogy justifies the choice of indicating the weighting factor as ν.

The results, for a network with 100 centres, are shown in figure 15. The plot shows the variations of the error on frame 2 and on frame 6 normalised to their self-result (that clearly occur at $\nu = 0$ for frame 2 and $\nu = 1$ for frame 6). The average result over the two frames is also shown.

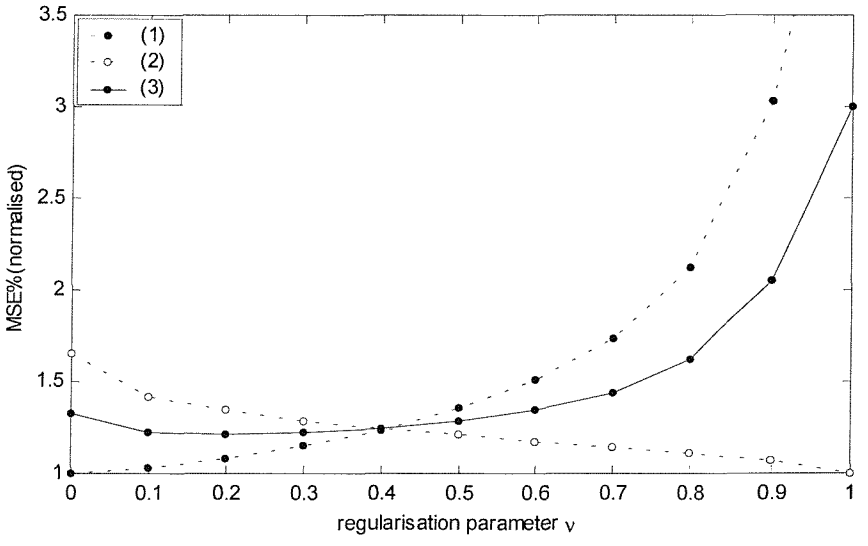

Figure 15. Extended training set. (1) result on frame 2 normalised on frame 2 self-result (2) result on frame 6 normalised on frame 6 self-result (3) average result.

From the average curve one may assume that, at its minimum, little improvement is achieved. However, from figure 16, where the variations are normalised to the corresponding linear self-result, one can see that for $\nu \approx 0.15$ there is a noticeable improvement for frame 6. Conversely, there is a much smaller decrease of performance for frame 2, to a level that might be considered acceptable. It is clear how the controlled extension of the training pattern can improve the performance of a network over particularly intractable inputs, at the same time assuring a satisfactory level of smoothness and generalisation.

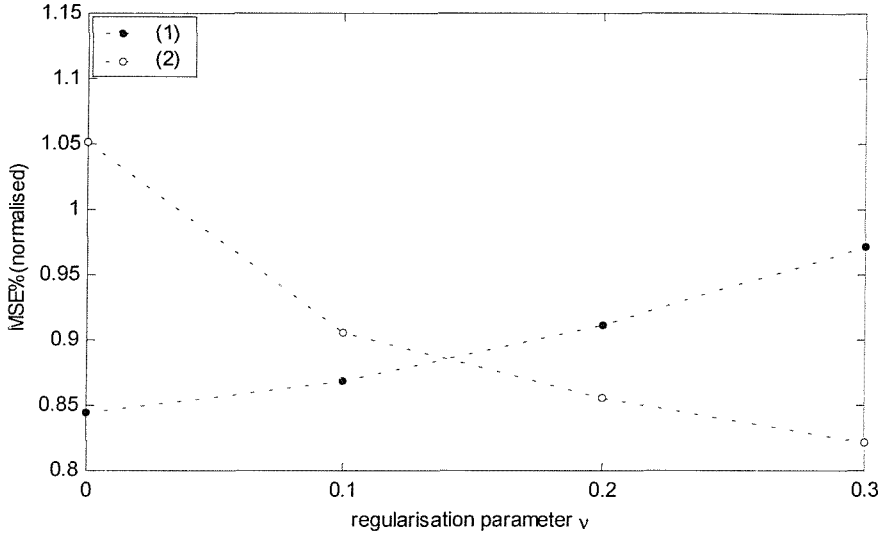

Figure 16. Extended training set. (1) result on frame 2 normalised on frame 2 linear self-result (2) result on frame 6 normalised on frame 6 linear self-result.

7.8. Comparison of results

The following figures show the comparative results obtained training the Volterra series and a 100-centres network on frame 2 using the different techniques described previously. In figure 17 one can see the MSE of the Volterra series 2 and RBFN network 2, unregularised and regularised, over the frame set. In the upper frame of the plot the results are normalised to the linear self-result, whilst the bottom plot is normalised with respect to the corresponding non-linear self-result. From these plots, it appears that the Volterra series produces a more general result than RBFN, albeit the difference is primarily on frame 5. However, the regularisation of the RBFN produces a result on frame 5 that is comparable to that obtained by the Volterra series.

That said in general, the application of regularisation does not produce any substantial advantage on the other frames.

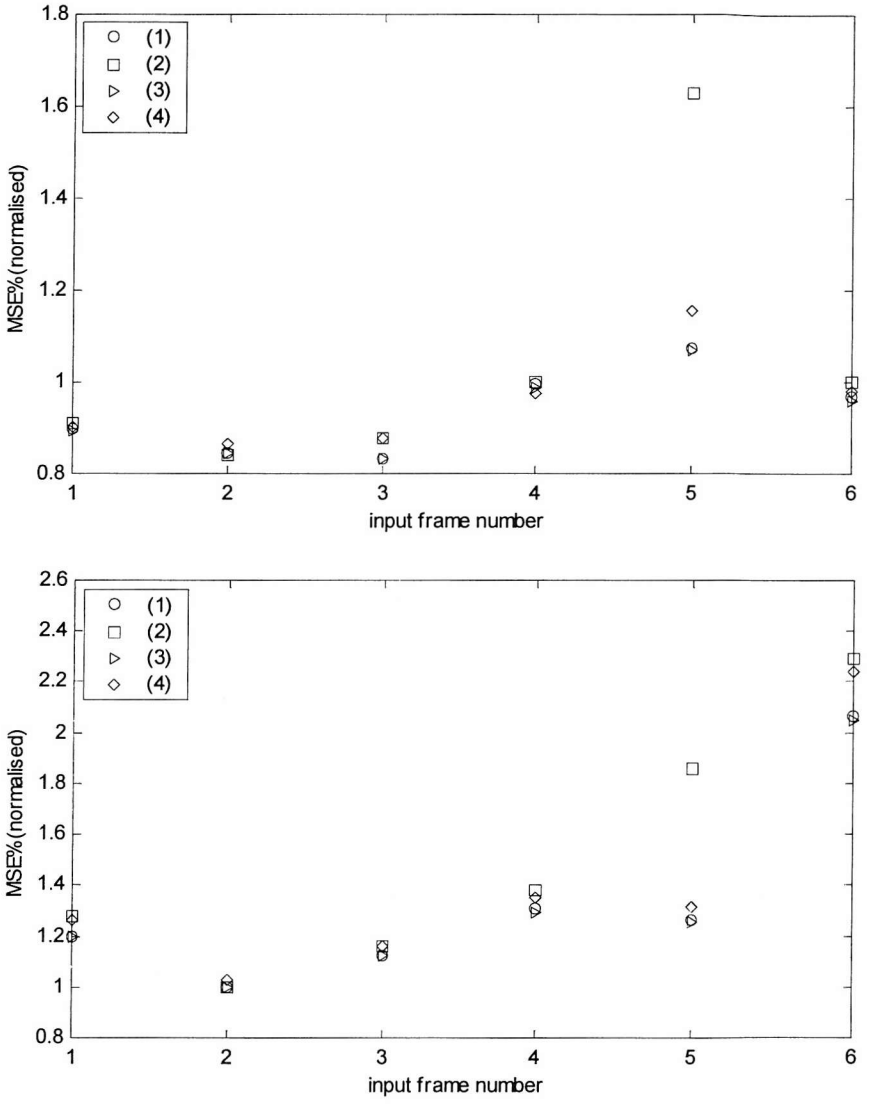

Figure 17. Comparison of Volterra and RBFN trained on frame 2, tested over the frame set. Top: normalised to linear self-result. Bottom: normalised to non-linear self-result. (1) Volterra series, (2) RBFN, (3) regularised Volterra series ($\nu = 10^{-4}$) (4) regularised RBFN ($\nu = 10^{-4}$).

Figure 18 shows the result of the application of the extended RBFN trained on frame 2 and 6, with $\nu = 0.15$. Note that the training is not regularised. The extended training has a beneficial effect and yields a MSE on frame 5 which is comparable to that obtained via regularisation, and with the expected improvement of the performance on frame 6.

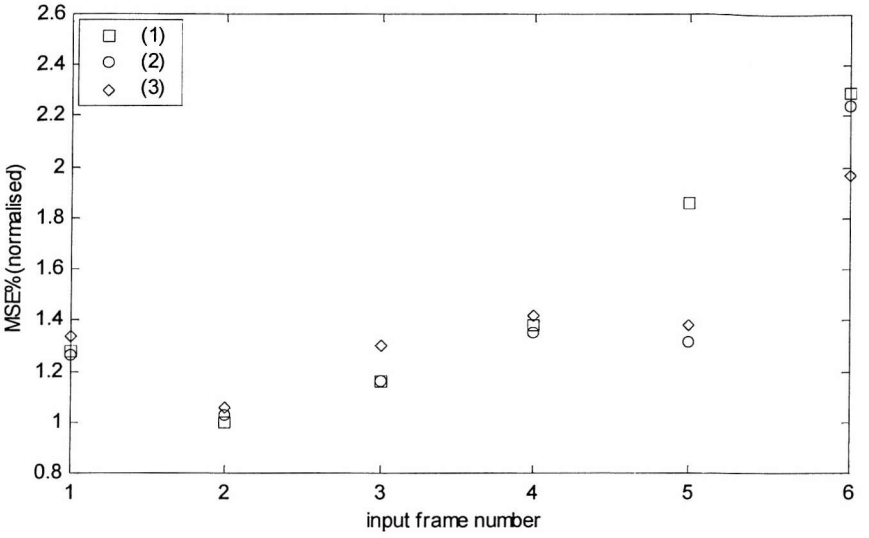

Figure 18. Extended training of RBFN network 2 (3), compared to non-regularised (1) and regularised training (2), normalised to self-result.

It is then reasonable to ask why frame 5 is so poorly de-interlaced by network 2, since the characteristics of the image do not look extremely different from frame 2 as in the case of frame 6. Figure 19 shows the frame 5 target field, and two error fields de-interlaced using RBFN. The first shows the absolute value of the difference between the network 5 self-result and the network 2 cross-result. In the second the difference is between the self-result and the cross-result of regularised network 2. It is clear that, among other differences, there is a large error concentrated on a large uniform white area (the collar), and that this error is basically a mismatch of the correct luminosity.

From the histograms in figure 2, note how frame 5 has a peak on the far right of the histogram, corresponding to a high density of white values. Such a peak is not present in the histogram of frame 2. It seems clear that the training on frame 2 fails to produce the correct output on uniform, white inputs, because there are not enough patterns in frame 2 to force a MSE solution to correctly account for these bright areas. Figure 20 shows this effect. The output of the de-interlacer is computed for inputs consisting of uniform input $\mathbf{x}(\alpha) = \alpha \cdot [1 \ldots 1]^T$ of value α, $\alpha \in [-128, 127]$. Figure 20 shows the output $y(\alpha)$ for a network trained on frame 2, a network trained on frame 5, and a network trained on frame 2 and regularised. The plot clearly shows that training on

frame 2 lacks precision in the right end of the range. Regularisation, by generally smoothing the response, tends to amend this problem.

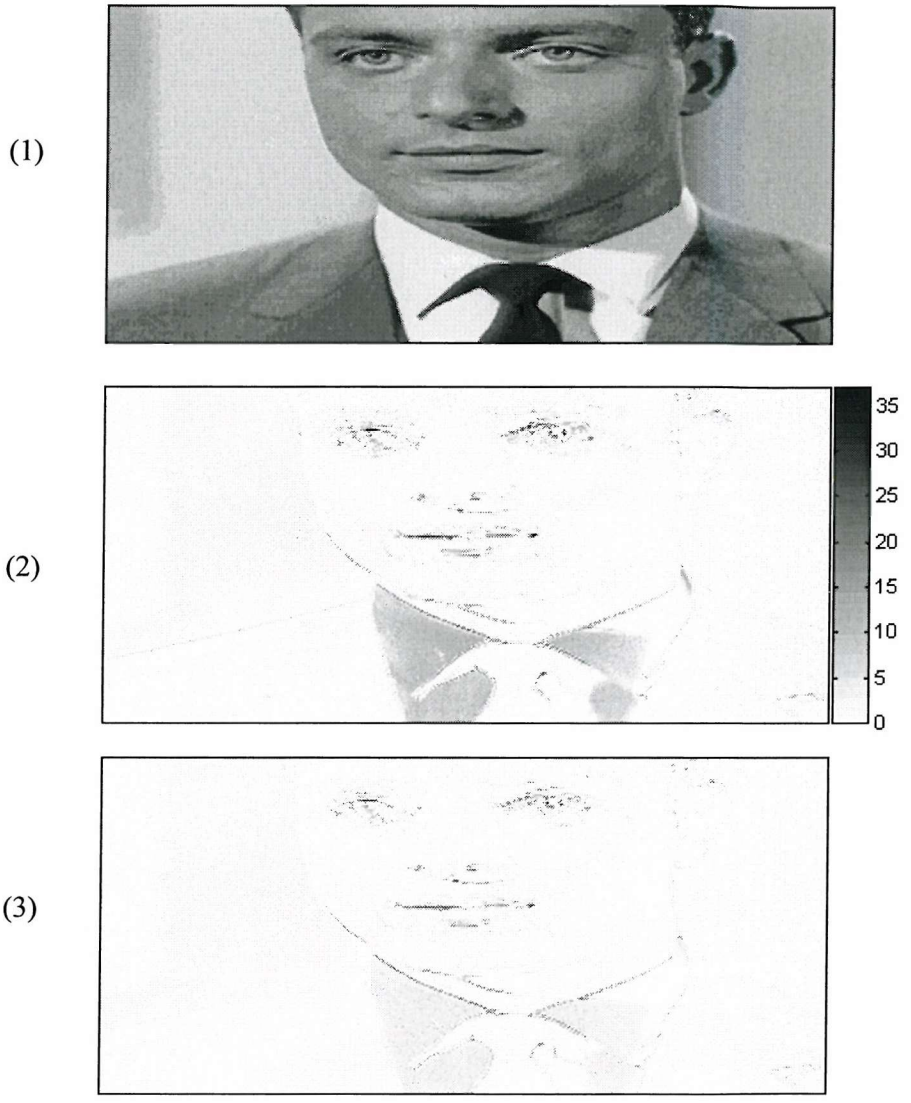

Figure 19. (1) field 5 interpolated by network 5. (2) Difference with same field interpolated by network 2 and (3) network 2 regularised.

The inset shows a detailed plot in the neighbourhood of $\alpha = 127$, allowing one to better appreciate the difference between the different results. The functions have been evaluated at 64 grey levels, which was chosen to correspond to the smallest perceptible grey level difference. Both the frame 5 self-result and the regularised training on frame 2 produce results whose tolerance is less than this difference, while the unregularised training on frame 2 fails this criterion.

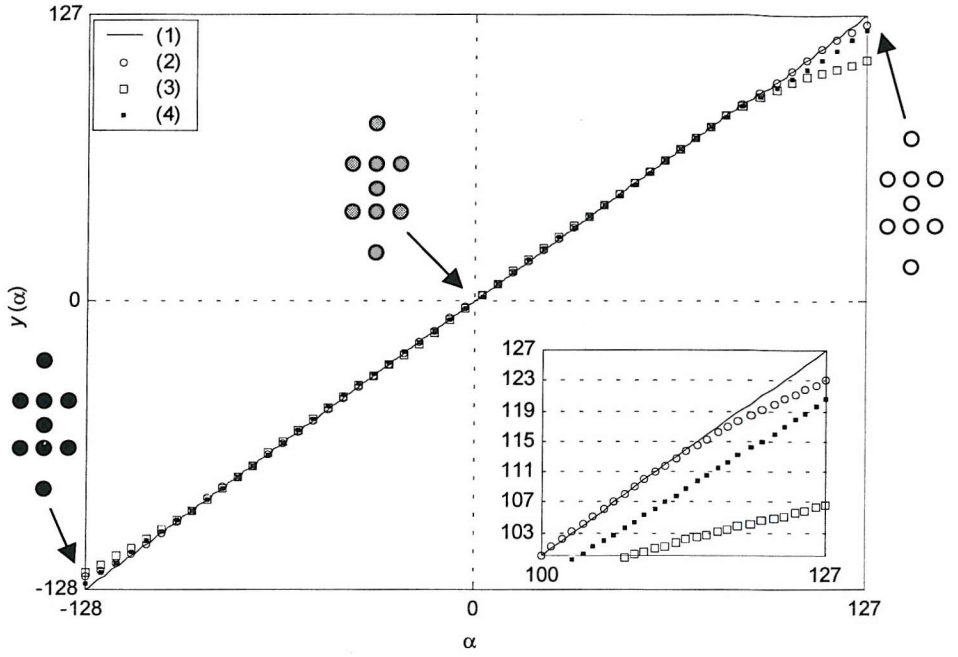

Figure 20. Network's response on the bisector of the input space. (1) ideal response (2) network 5 self-result (3) network 2 self-result (4) network 2 regularised. Three of the corresponding input-output pattern lattices are also shown.

7.9. Constrained optimisation

The previous section showed that a network trained on frame 2 fails to generate a proper mapping of patterns on the diagonal of the input space. This produces a perceputally disturbing artefact, because incorrect outputs are placed in a regular geometric pattern (field lines). An example of this is in figure 21.

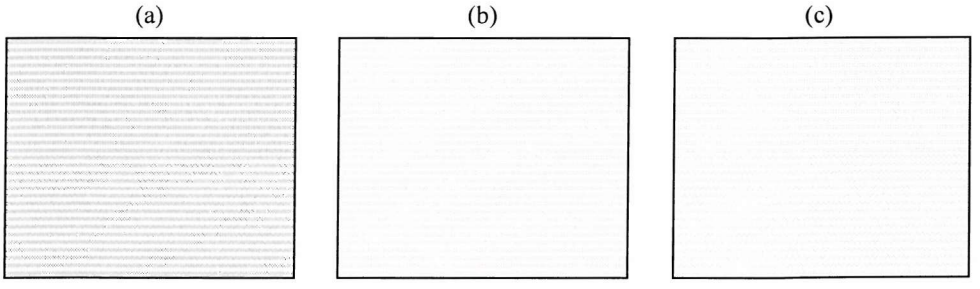

Figure 21. Line artefacts. Uniform white area de-interlaced using (a) network trained on frame 2 (b) network trained on frame 2, regularised (c) network trained on frame 5. Images are enlarged 2 times for a clearer perception of the artefacts.

For these patterns, which represent uniform areas in the image, the most desirable outcome is known a priori. However, this information is not explicitly provided to the

training procedure. Broadly speaking, if the training sequence includes a sufficient number of these patterns, like in frame 5, the result will generate a correct mapping along the diagonal. If this information is absent, like in frame 2 for $\alpha \approx 127$, then the training procedure may produce incorrect results. It seems reasonable to build a network that has an explicit ability to map these patterns, regardless their relative preponderance in the training set. A way to achieve this is by constraining the optimisation to satisfying these conditions.

To achieve this the optimisation is modified by incorporating a linear constraint. Specifically the problem becomes that of minimising (7.7) subject to the constraint

$$\mathbf{\Phi}^T \mathbf{h}_\alpha - \alpha = 0 \quad \forall \alpha \in [-128, 127] \tag{7.13}$$

where $\mathbf{h}_\alpha$ is the vector of nodal outputs assuming a uniform input with grey level α. In practice a limited number of constraints are used. One should understand that by imposing the constraints on (7.7) the solution departs from the MMSE solution. An excessive number of constraints lead to an inconsistent solution, i.e. there might not be a minimiser for the constrained problem. Supposing that there are K constraints, $\alpha \in [\alpha_1 \ldots \alpha_K]$, (7.13) can be expressed in matrix form:

$$\begin{aligned} &\mathbf{K}^T \mathbf{\Phi} = \mathbf{A} \\ &\mathbf{K} = [\mathbf{h}_{\alpha_1} \ldots \mathbf{h}_{\alpha_K}] \\ &\mathbf{A} = [\alpha_1 \ldots \alpha_K]^T \end{aligned} \tag{7.14}$$

7.9.1. Lagrange multipliers

A well known technique to obtain constrained optimisation is by Lagrange multipliers (Fletcher, 1987). The solution to the constrained optimisation is obtained by minimising the *Lagrangian function* with respect to $\mathbf{\Phi}$ and $\mathbf{\Lambda}$

$$C(\Phi) = \Phi^T \mathbf{R}\, \Phi - 2 \cdot \Phi^T \mathbf{P} - \Lambda^T \left(\mathbf{K}^T \Phi - \mathbf{A} \right) \tag{7.15}$$

where $\Lambda = [\lambda_1 \ldots \lambda_K]^T$ is the set of Lagrange multipliers, one for each constraint. If the inverse of the *Lagrangian matrix* **L** exists,

$$\mathbf{L}^{-1} = \begin{bmatrix} \mathbf{R} & -\mathbf{K} \\ -\mathbf{K}^T & \mathbf{0} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{B} & -\mathbf{T} \\ -\mathbf{T}^T & \mathbf{U} \end{bmatrix} \tag{7.16}$$

then the solution in Φ can be written as:

$$\Phi = \mathbf{B}\mathbf{P} + \mathbf{T}\Lambda \tag{7.17}$$

The explicit expressions for **B** and **T** are:

$$\begin{aligned} \mathbf{B} &= \mathbf{R}^{-1} - \mathbf{R}^{-1}\mathbf{K}\left(\mathbf{K}^T\mathbf{R}^{-1}\mathbf{K}\right)^{-1}\mathbf{K}^T\mathbf{R}^{-1} \\ \mathbf{T} &= \mathbf{R}^{-1}\mathbf{K}\left(\mathbf{K}^T\mathbf{R}^{-1}\mathbf{K}\right)^{-1} \end{aligned} \tag{7.18}$$

In figure 22 we show the results on the diagonal of the input space, for 3 constraints $\alpha = 0$, $\alpha = -128, +127$, (corresponding to the extremes and the mid-point of the diagonal), for 5 constraints (additional constraints at $\alpha = \pm 64$), and for 9 constraints (additional constraints at $\alpha = \pm 32, \alpha = \pm 96$). It is clear how, as the number of constraints increases, the performance on the diagonal increases. In fact the result for 5 constraints is nearly indistinguishable from those obtained with 9 constraints. In figure 23 we show the normalised cross-results for the unconstrained and constrained cases. One can see the noticeable improvement in the results for frame 5. Again, there is a negligible difference among the three constrained schemes.

More constraints could be considered by investigating other a-priori patterns. One could use inequalities in (7.13) rather than equalities (e.g., $\Phi^T \mathbf{h}_\alpha - \alpha \leq 1$, if we

intend to use 8-bit precision in the range -128 to 127). That would afford more flexibility to the optimisation algorithm. However, care should be taken to abide by the main goal of the optimisation (i.e. the minimisation of the error), and avoid inconsistent constraints.

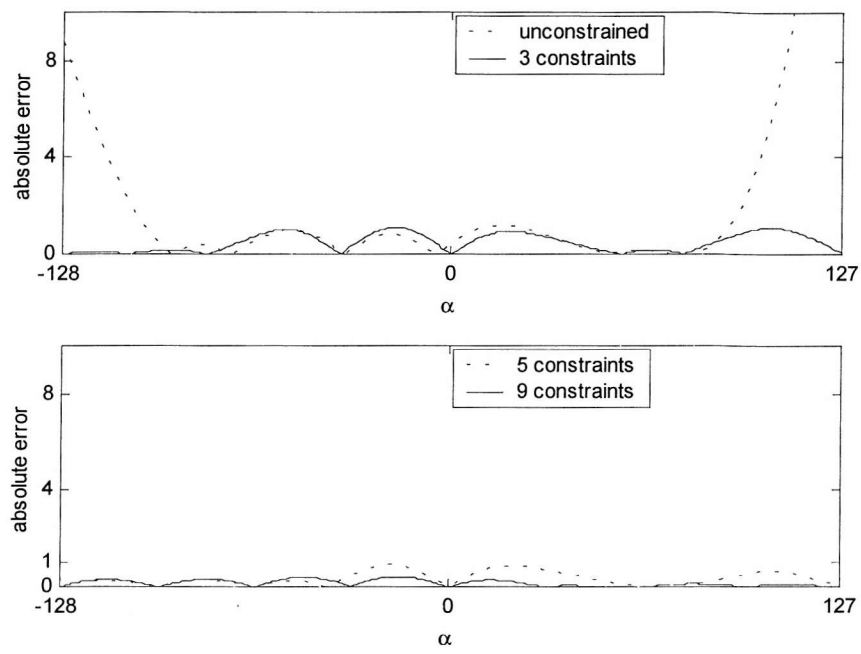

Figure 22. Constrained optimisation of network 2, 100 centres: absolute error on the diagonal.

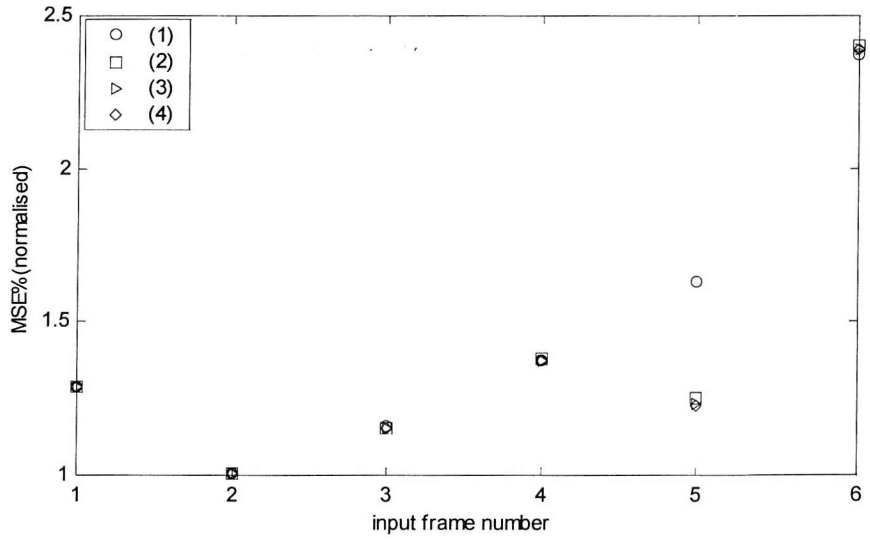

Figure 23. Cross-results, network 2 with 100 centres. (1) unconstrained (2) 3 constraints (3) 5 constraints (4) 9 constraints.

7.10. Mixture of experts

The difficulty in reproducing the self-result on frame 6 without directly training on it has been demonstrated. The extended training of network 2 on frames 2 and 6 reduces the gap, albeit not completely. It has also been shown that, with the proper choice of centres, the burden of specialisation falls mainly on the network's weights.

This raises the question as to whether a network with fixed weights is an appropriate solution to the problem. One may imagine a model that changes its structure according to the pattern presented at its input. A possible solution would be to create a set of K networks that specialise their mappings on different sectors of the input space. These networks might be even use different architectures, e.g. being a mixture of linear, RBFN, and Volterra networks. A gating network selects the expert network, i.e. the best mapping among the different alternatives, according to the position of the current input in the input space. Such a structure is known as mixture-of-experts model (Jacobs *et al.*, 1991). The schematic diagram of the model is depicted in figure 24.

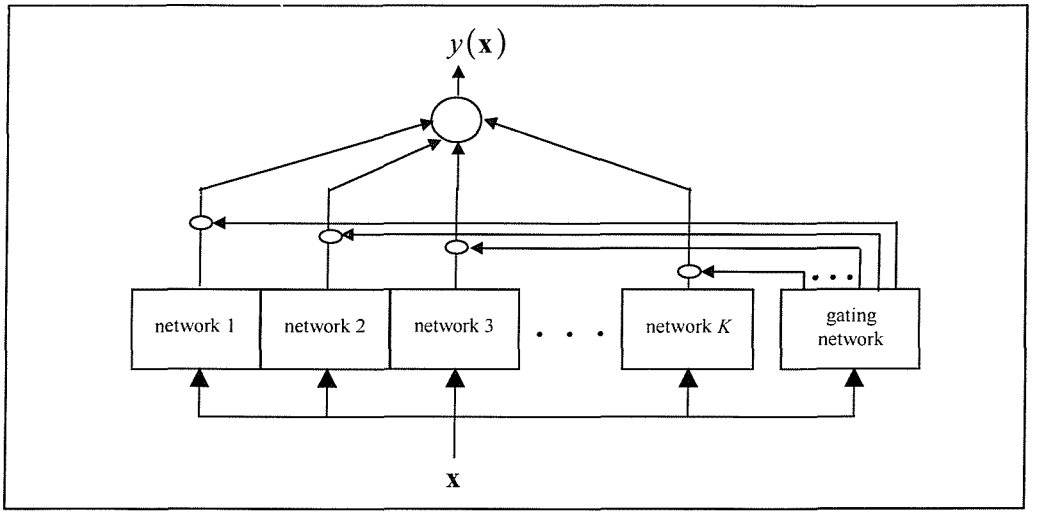

Figure 24. Schematic diagram of a mixture of experts model.

This model can be described by the equation:

$$y(\mathbf{x}) = \sum_{j=1}^{K} \alpha(\gamma_j) y_j(\mathbf{x}) \tag{7.19}$$

where $y_j(\mathbf{x})$ is the output of the network j, $\gamma_j = \gamma_j(\mathbf{x})$ is the j-th output of the gating network, and $\alpha(\gamma_j)$ is the mixing coefficient j. α_j can be either a "hard" switching function, in which case only one network operates at any time, or a "soft" switching function, in which case the output is a weighted sum of all the networks' responses. The latter solution is preferable, since a properly trained system guarantees continuity and smoothness on the transition from one partition of the input space to another. This avoids artefacts due to responses on similar patterns that lay across a partition boundary. An example of a soft gating is given by the *softmax* function (Bridle, 1990; Jacobs, 1991):

$$\alpha(\gamma_j) = \frac{exp(\gamma_j)}{\sum_{k}^{K} exp(\gamma_k)} \tag{7.20}$$

Another point in favour of a "soft" switching function is that it can also be demonstrated (Bishop, 1995) that the error of a weighted sum of networks can be singnificantly reduced. A weighted sum of networks is referred in literature as *committee of networks* (Perrone and Cooper, 1993; Perrone, 1994). This reduction can be seen as arising from a reduced variance due to the averaging over many networks. Therefore, it is suggested (Bishop, 1995) that the individual networks should not optimally trade-off bias and variance, but rather prefer smaller bias, since the variance is reduced by the averaging process.

To illustrate the applicability of the technique to the de-interlacing problem, a system is developed based on separating the input into high and low contrast regions. Two

features are derived for the input vector **x**. One is the projection $x_{\|}$ of **x** onto the diagonal, and the other is the norm of the component of **x** orthogonal to the diagonal, that we call $x_{\perp} = \|\mathbf{x}_{\perp}\|$, see figure 25. The two axis are denoted as π (parallel) and ω (orthogonal). The value of $x_{\perp}$ can be interpreted as a measure of contrast, whilst $x_{\|}$ is a measure of brightness. It is then relatively easy to decompose the input space in a high-contrast and a low-contrast region. Figure 26 shows the squared errors resulting from applying various de-interlacing schemes to the whole frame set, plotted in the π-ω plane. The errors are normalised by the total sum of squared errors. Results are shown for a network fully trained on frame 2, for a network using centres from frame 2 and the weights recalculated on frame 6, and a network fully trained on frame 6. White areas represent zones where no input has been encountered.

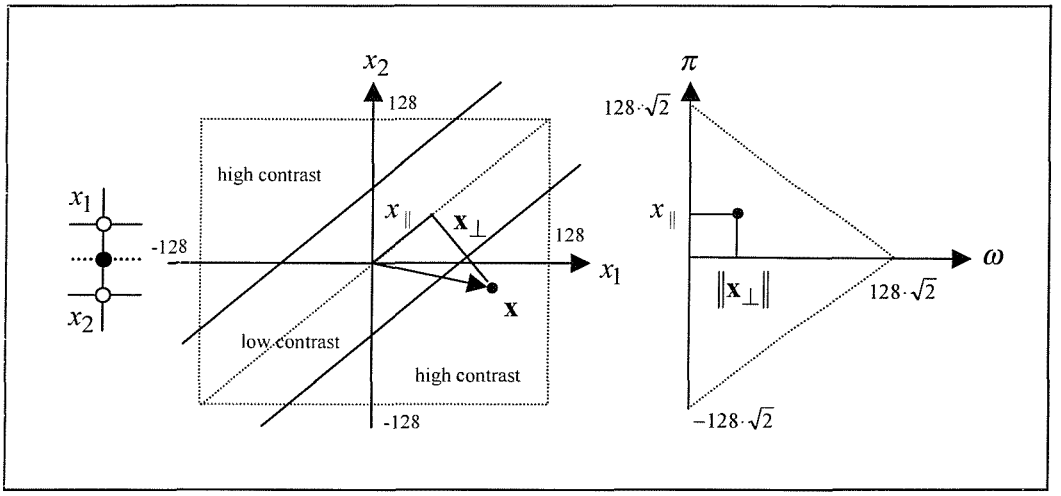

Figure 25. Parallel and orthogonal components illustrated for a 2-tap vertical lattice.

These differences can be further appreciated in figure 27. The difference in the π-ω plane between the errors of the network trained on frame 6 and of the network trained on frame 2 is plotted, along with the difference between the errors of the network trained on frame 6 and the network initially trained on network 2 and with calculated on frame 6. A negative value indicates that in that area network 6 produces on average a smaller error.

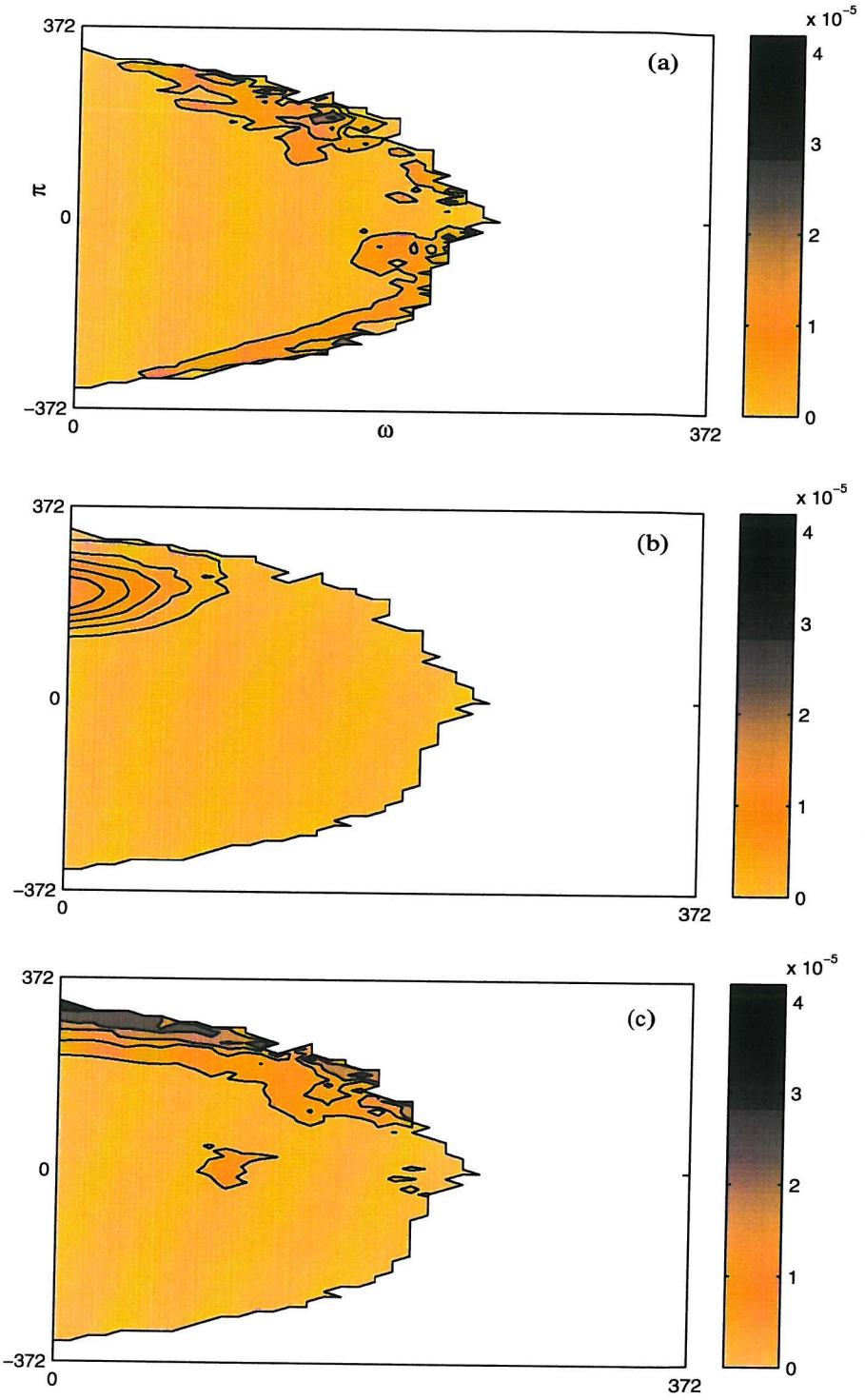

Figure 26. ω-π error planes, showing the average error calculated using the full frame set. Network 2 (a), network 2 with weight calculation on frame 6(b), network 6 (c).

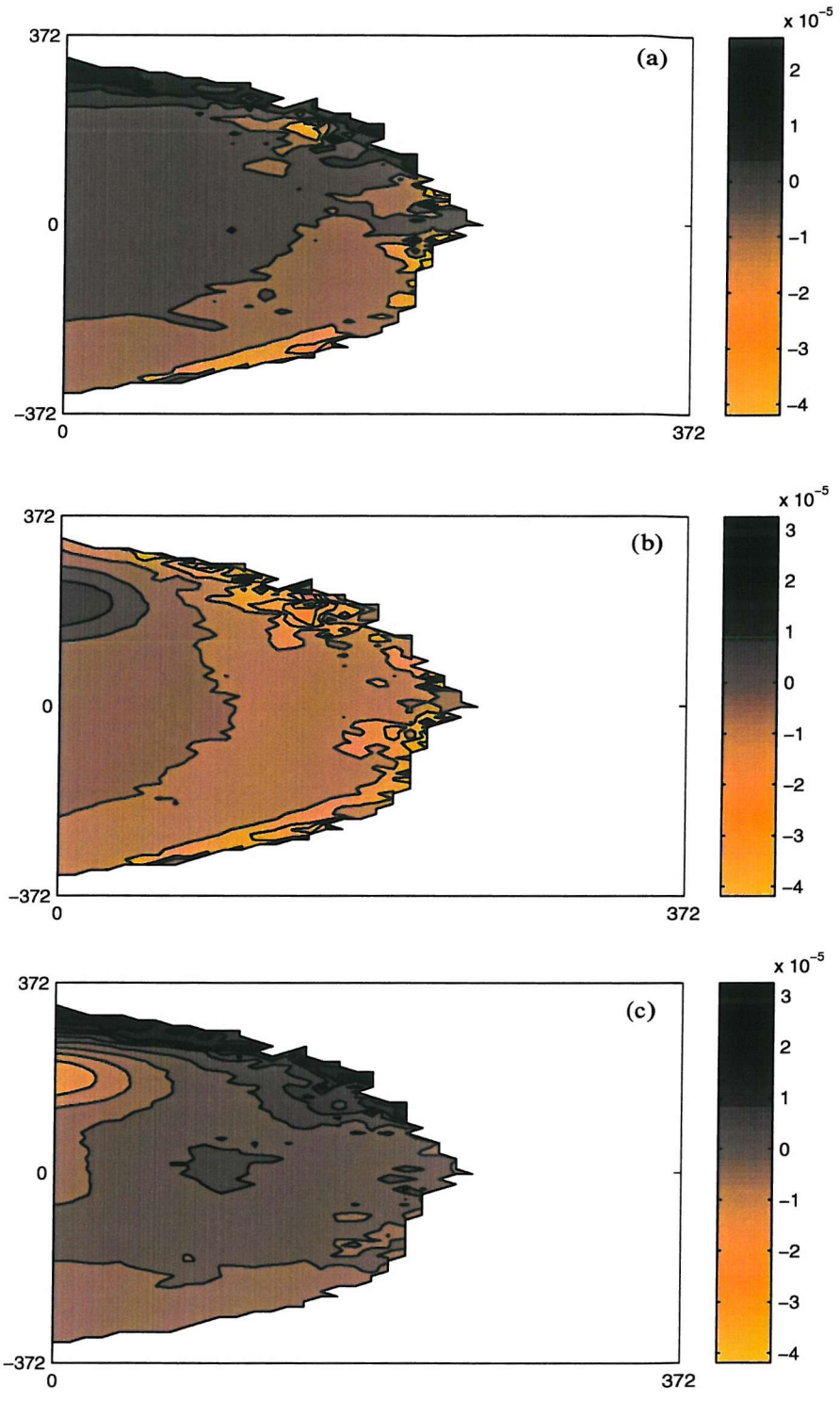

Figure 27. ω-π error difference planes, showing the average error difference between different training schemes, calculated using the full frame set. Network 6 minus network 2 (a), network 6 minus network 2 with weight calculation on frame 6 (b), network 2 with weight calculation on frame 6 minus network 2 (c).

These figures illustrate that networks 2 and 6 perform well in different regions of the π-ω plane. Note also how network 2, with weights calculated on frame 6, has a smoother performance than the other two networks. Therefore, a mixture of experts, i.e. a system that applies different de-interlacing schemes to different regions of the π-ω plane, could produce a result that is general but also able to produce good results on difficult frames like frame 6.

It is beyond the scope of this thesis to investigate this possibility. What it is suggested here is that the π and ω components of $\mathbf{x}$ could be used by the gating network to switch between a general de-interlacer like a RBFN trained on frame 2 and constrained on the diagonal, and a more specialised system like a RBFN trained on frame 6.

7.11. Conclusions

In chapter 5 and 6 the linear filter, the Volterra series and RBFN have been investigated in order to realise a de-interlacing system. The results have shown that non-linear techniques have superior performance in terms of the resulting MSE. However, in this section it has been shown how this increased performance pays the cost of reduced generalisation ability. The better the performance achieved on a single frame is, the worse the performance achieved on frames not included in the training set is. Including additional patterns (i.e. frames) in the training set is not a feasible solution, since it increases the computational load.

Several theoretical aspects of generalisation have been investigated. It has been seen how the MSE can be decomposed into a component that depends on the lack of complexity of the model (bias), and a component that depends on the specialisation on the training set considered (variance).

The main problem is that the unconstrained minimisation of the MSE leads to excessive specialisation. Therefore the lack of generalisation lays mainly in the training procedure. It is necessary to devise a cost function that, along with the error term, includes a term that measures the generalisation ability of the system and trades-off the minimisation of the MSE. This approach is known as regularisation.

One possible form of regularisation is the weight-decay training, often referred as ridge-regression. The principle is that a specialised network is non-smooth, and that non-smooth mappings yield high weight values. Weight-decay seeks generality by trading-off the MSE with the sum of the squared weights. The results show that this technique has a beneficial effect on non-general systems.

Another possible technique to add generality is to introduce prior constraints to the training. For many input patterns the input can be easily forecast. By constraining the minimisation of the MSE to consider these known patterns, the system achieves generality since this prior knowledge is derived from general considerations on images. The results show that the generality of the system is increased, without significantly affecting the MSE performance.

However, some patterns present peculiar characteristics that are rarely found in a typical training set. An example is a frame containing a page of text. A network trained on this frame yields poor results on other more "general" frames. Conversely, a network trained on a general training set cannot achieve the good performance of a specialised system. A solution could be to include some of these patterns in the training set. The results show that some improvements can be achieved in this way.

A better solution is to create a network that changes its mapping according to the input. The investigation of such a network is beyond the scope of this thesis. Nevertheless, a simple introduction to the subject is given, together with a suggested technique to determine the appropriate system's mapping according to some properties of the input in the input space.

In the next two chapters, the subject of symmetry is discussed for linear filters and RBFN. It will be shown that, assuming rotational invariance for the mapping, it is possible to devise prior conditions that lead to a reduced computational load. In chapter 10 it will be shown that this also leads to an increased generality of the system.

8. SYMMETRY CONSTRAINTS FOR LINEAR NETWORKS

8.1. Introduction

In many practical situations, the input space created by the process of sampling contains information that is, in some way, redundant. The removal of any redundancy may reduce the dimensionality of the problem and result in a reduction of the computational effort. By means of simple operations, typically linear ones, it is often possible to transform the input space into a new space that has a lower dimension. The subsequent application of non-linear techniques clearly benefits from such a reduction.

Sometimes such reduction can be obtained in an intuitive way. As a simple example, if the system task is to process the average luminosity of the input vector, it is wise to calculate this quantity first by a simple weighted sum, so that the dimensionality is reduced to 1, and then process the result. In the literature, such process is often referred as *feature extraction.* In other words, one tries to obtain the salient information (features) from the input space, so that this information has a reduced dimensionality compared to the original space. The process of identifying the features of interest requires an analysis of the problem that is both analytical and intuitive.

This chapter exploits a particular feature set derived for images that applies whenever the sampling lattice has certain geometrical symmetries. Consider the picture in figure 1, where a square neighbourhood of the pixel p is sampled to obtain an input vector $\mathbf{x}$. Now, suppose the image is rotated around p by 90 degrees; in general the input will be different and denoted $\mathbf{x}_{90}$. Clearly, the target pixel's value p has not changed, and more importantly the geometry of the lattice in relation to the image has not changed either, since the lattice can be rotated by any integer multiple of 90°. It is therefore desirable that the interpolator produces the same output when either $\mathbf{x}$ or $\mathbf{x}_{90}$ are presented to the input.

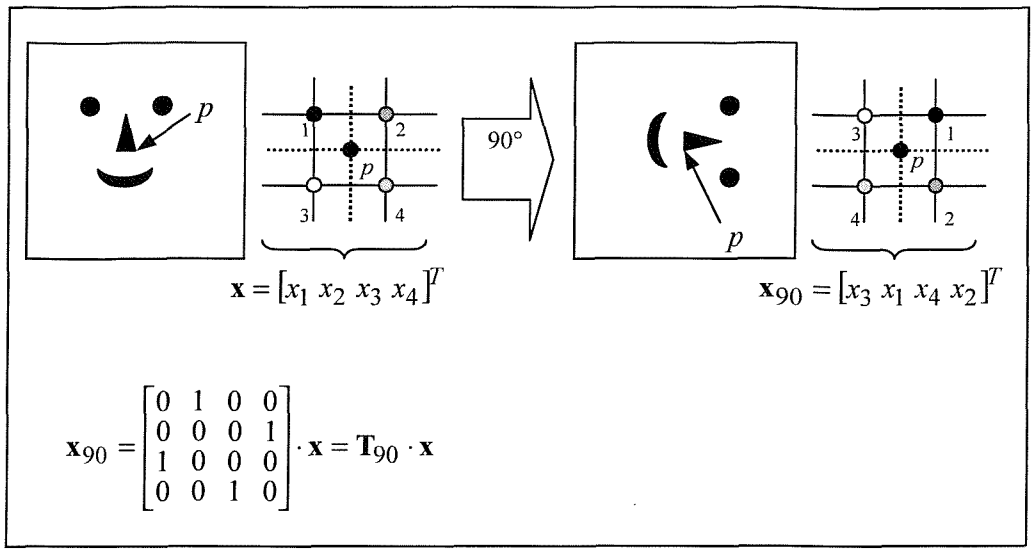

Figure 1. Symmetry in image sampling.

More formally, if $f(\mathbf{x})$ is the interpolation function, then it is required that:

$$f(\mathbf{x}) = f(\mathbf{x}_{90}), \forall \mathbf{x} \tag{8.1}$$

This equation is called a *condition of symmetry*, in the sense that it describes a condition any interpolator should respect, regardless the particular image/training set considered, when the sampling lattice is transformed according to its symmetries. It is then an a priori constraint, valid for all images. Note that we are able to draw a matrix relation between the two inputs, $\mathbf{x}$ and $\mathbf{x}_{90}$. The matrix $\mathbf{T}_{90}$ (figure 1) is then called a rotation operator, with the following properties:

- The matrix $\mathbf{T}_{90}$ is full rank, $rank(\mathbf{T}_{90}) = 4$.
- The columns of $\mathbf{T}_{90}$ are equal to the columns of the identity matrix, rearranged.
- The columns of the matrix $\mathbf{T}_{90}$ form an orthonormal basis in R^4.
- $\|\mathbf{x}\| = \|\mathbf{T}_{90}\ \mathbf{x}\|$, $\|\mathbf{x}_1 - \mathbf{x}_2\| = \|\mathbf{T}_{90}\ \mathbf{x}_1 - \mathbf{T}_{90}\ \mathbf{x}_2\|$ (Isometry)

Note that all these properties are intimately linked. They are consequences of the fact that the operator $\mathbf{T}_{90}$ simply swaps the position of pixels associated in a one-to-one fashion. More generally, one can say that $\mathbf{T}_{90}$ is a member of a class of symmetric operators on the sampling lattice. These operators transform the input space into another with same span (isometry), with the constraint that the operator is derived from a symmetric transform of the sampling lattice.

8.2. Symmetric transforms of the sampling lattice

$\mathbf{T}_{90}$ is not the only possible transform of the sampling lattice shown in figure 1 that preserves the sampling topology (symmetry). The chosen lattice, being a square, allows 8 possible transformations, with the same topological/algebraic properties as $\mathbf{T}_{90}$ (i.e. being a rearrangement of the co-ordinates system, and preserving the geometry). Figure 2 illustrates all these transforms. Following the previous considerations, all the input vectors coming from these rotations should lead to the same interpolator output.

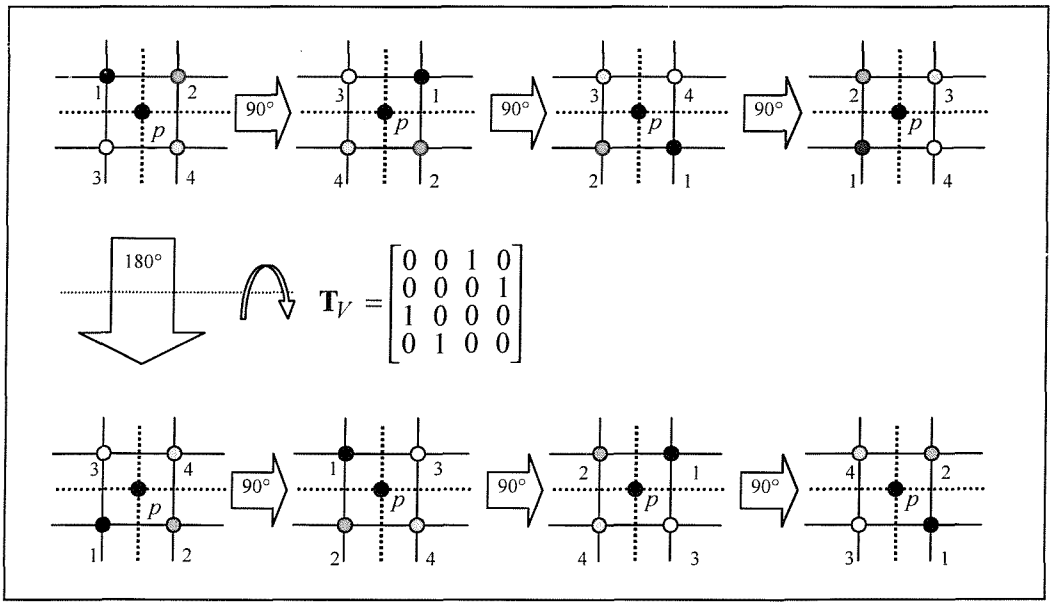

Figure 2. Self-similar transforms of the square lattice.

Note the introduction of a new operator $\mathbf{T}_V$ that performs the rotation about the horizontal axis (mirror symmetry), $\mathbf{T}_V$ shall be referred to as the "vertical flip"

operator, since it "flips" the pixels in the vertical direction. The operator $\mathbf{T}_V$ has the same properties as $\mathbf{T}_{90}$, and belongs to the same class of operators.

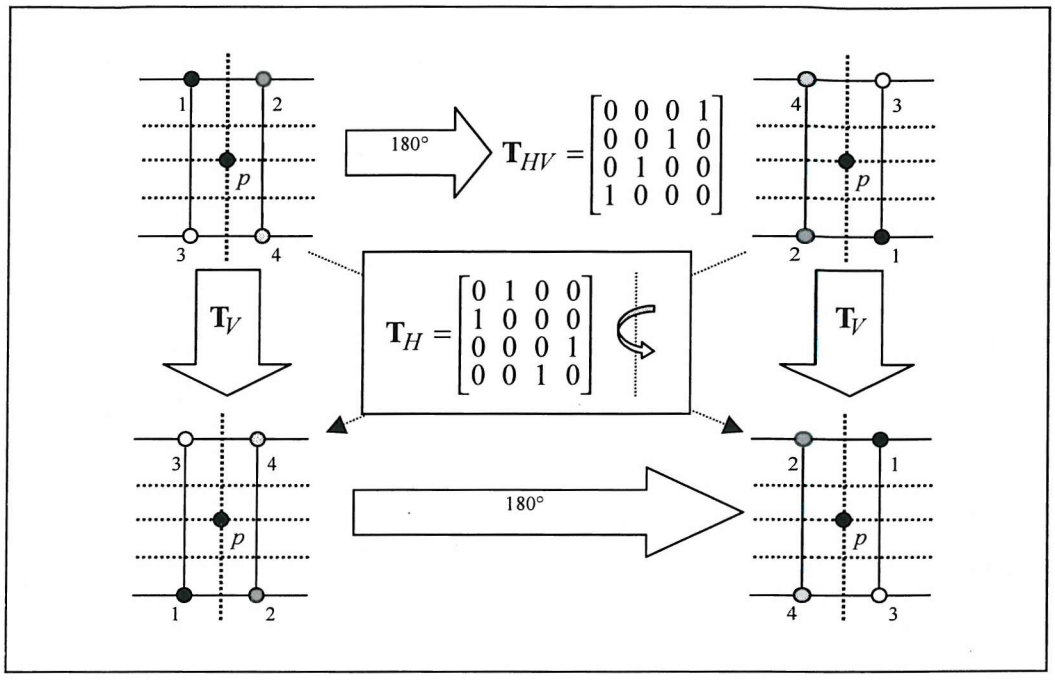

Figure 3. Self-similar transforms of the rectangular lattice.

8.2.1. Symmetric transforms of a rectangular lattice

The picture described for the square sampling lattice alters if the shape is rectangular (figure 3). The problem is that not all the rotations shown for the square lattice preserve the shape of the rectangular lattice. Specifically, the $\mathbf{T}_{90}$ transform cannot be applied. In the rectangular case the number of transformations is smaller than in the case of square sampling lattice, as can be seen in figure 3. Two new transforms, $\mathbf{T}_H$ (horizontal flip) and $\mathbf{T}_{HV}$ (horizontal-vertical flip), have been introduced. Since the only transforms allowed are 180° degree rotations, one can see that a rotation followed by a vertical flip is equivalent to a flip about the vertical axis, that is expressed by $\mathbf{T}_H$. The transform $\mathbf{T}_{HV}$ has the following properties:

$$\mathbf{T}_{HV} = \mathbf{T}_{90}\,\mathbf{T}_{90} \tag{8.2.a}$$

$$\mathbf{T}_{HV} = \mathbf{T}_V\,\mathbf{T}_H = \mathbf{T}_H\,\mathbf{T}_V \tag{8.2.b}$$

8.2.2. Conditions of symmetry

The conditions of symmetry for the two cases can be derived:

- Square sampling (4 pixels):

$$\forall \mathbf{x} \begin{cases} f(\mathbf{x}) = f(\mathbf{T}_{90}\,\mathbf{x}) \\ f(\mathbf{x}) = f(\mathbf{T}_{HV}\,\mathbf{x}) = f(\mathbf{T}_H\,\mathbf{T}_V\,\mathbf{x}) = f(\mathbf{T}_V\,\mathbf{T}_H\,\mathbf{x}) \\ f(\mathbf{x}) = f(\mathbf{T}_H\,\mathbf{x}) \\ f(\mathbf{x}) = f(\mathbf{T}_V\,\mathbf{x}) \end{cases} \tag{8.3.a}$$

- rectangular sampling (4 pixels):

$$\forall \mathbf{x} \begin{cases} f(\mathbf{x}) = f(\mathbf{T}_{HV}\,\mathbf{x}) = f(\mathbf{T}_H\,\mathbf{T}_V\,\mathbf{x}) = f(\mathbf{T}_V\,\mathbf{T}_H\,\mathbf{x}) \\ f(\mathbf{x}) = f(\mathbf{T}_H\,\mathbf{x}) \\ f(\mathbf{x}) = f(\mathbf{T}_V\,\mathbf{x}) \end{cases} \tag{8.3.b}$$

In general, any combination (multiplication) of the operators must satisfy the condition of symmetry, that is:

$$\forall \mathbf{x} : f\left[\left(\prod_{m=1}^{M} \mathbf{T}_m\right)\mathbf{x}\right] = f\left[\left(\prod_{n=1}^{N} \mathbf{T}_n\right)\mathbf{x}\right] \tag{8.3.c}$$

where the set of operators $\{\mathbf{T}_m\}, m = 1...M$, $\{\mathbf{T}_n\}, n = 1...N$ are drawn from the set of all the possible operators for the given lattice.

8.3. Symmetric reduction of the input space – the linear case

In this section, it will be shown how the symmetry properties of the sampling lattice can be exploited when the interpolator is linear. The case of a linear interpolator is initially addressed to provide foundations for the extension to the non-linear case. This is based on extracting features from the input space that are invariant under the transforms $\mathbf{T}_{90}$, $\mathbf{T}_H$, $\mathbf{T}_V$ and $\mathbf{T}_{HV}$.

In the case of linear systems it is easy to deal with the algebra of the conditions of symmetry. We shall present in detail the case of a 4 pixel square aperture as an example of the methodology adopted. The linear interpolator can be expressed as:

$$f(\mathbf{x}) = \mathbf{\Phi}^T \mathbf{x}, \quad \mathbf{\Phi} = [\phi_1 \ \phi_2 \ \phi_3 \ \phi_4]^T \tag{8.4}$$

Applying the conditions (8.3.a) (square shape) to (8.4), yields:

$$\forall \mathbf{x} \quad \begin{cases} \mathbf{\Phi}^T (\mathbf{I} - \mathbf{T}_{90})\mathbf{x} = \mathbf{0} \\ \mathbf{\Phi}^T (\mathbf{I} - \mathbf{T}_{HV})\mathbf{x} = \mathbf{0} \\ \mathbf{\Phi}^T (\mathbf{I} - \mathbf{T}_{H})\mathbf{x} = \mathbf{0} \\ \mathbf{\Phi}^T (\mathbf{I} - \mathbf{T}_{V})\mathbf{x} = \mathbf{0} \end{cases} \tag{8.5}$$

where $\mathbf{I}$ and $\mathbf{0}$ are the identity matrix and the null vector in R^4. It follows that $\mathbf{\Phi}$ should be orthogonal to the spaces defined by the span of $(\mathbf{I} - \mathbf{T}_{90})$, $(\mathbf{I} - \mathbf{T}_{HV})$, $(\mathbf{I} - \mathbf{T}_{H})$ and $(\mathbf{I} - \mathbf{T}_{V})$. It is easy to see that:

$$rank\{\mathbf{I} - \mathbf{T}_i\} = 2, \ \forall \mathbf{T}_i \in \{\mathbf{T}_{90}, \mathbf{T}_H, \mathbf{T}_V, \mathbf{T}_{HV}\} \tag{8.6}$$

Hence $(\mathbf{I} - \mathbf{T}_i)$ is 2-dimensional and, considering that the input space is 4-dimensional, it follows that the 4 spaces described in (8.5) can not be orthogonal. The simplest way to calculate the combined dimension of these spaces is to compute the rank of the matrix product of the operators, i.e. the dimensionality of the projection of one space onto the others. This leads to the following results:

$$\left.\begin{array}{l} rank\left[(\mathbf{I} - \mathbf{T}_H)^T (\mathbf{I} - \mathbf{T}_V)\right] \\ rank\left[(\mathbf{I} - \mathbf{T}_V)^T (\mathbf{I} - \mathbf{T}_{HV})\right] \\ rank\left[(\mathbf{I} - \mathbf{T}_H)^T (\mathbf{I} - \mathbf{T}_{HV})\right] \end{array}\right\} = 1 \tag{8.7a}$$

Hence, $(\mathbf{I}-\mathbf{T}_V),(\mathbf{I}-\mathbf{T}_H)$ and $(\mathbf{I}-\mathbf{T}_H\mathbf{T}_V)$ overlap each other in one dimension. Furthermore

$$\left.\begin{array}{l} rank\left[(\mathbf{I}-\mathbf{T}_H)^T(\mathbf{I}-\mathbf{T}_{90})\right] \\ rank\left[(\mathbf{I}-\mathbf{T}_V)^T(\mathbf{I}-\mathbf{T}_{90})\right] \\ rank\left[(\mathbf{I}-\mathbf{T}_{HV})^T(\mathbf{I}-\mathbf{T}_{90})\right] \end{array}\right\} = 2 \qquad (8.7b)$$

Therefore one can see that from (8.6) the 2-D space $(\mathbf{I}-\mathbf{T}_{90})$ is equivalent to $(\mathbf{I}-\mathbf{T}_{HV})$, $(\mathbf{I}-\mathbf{T}_H)$ and $(\mathbf{I}-\mathbf{T}_V)$. Considering that $\mathbf{T}_i=\mathbf{T}_i^T, i=H,V,HV$, we also have:

$$\begin{array}{l} \mathbf{I}-\mathbf{T}_{HV}=\mathbf{T}_H\mathbf{T}_H-\mathbf{T}_H\mathbf{T}_V=\mathbf{T}_H(\mathbf{T}_H-\mathbf{T}_V)= \\ \mathbf{T}_H(\mathbf{T}_H-\mathbf{I}+\mathbf{I}-\mathbf{T}_V)=\mathbf{T}_H[(\mathbf{I}-\mathbf{T}_V)-(\mathbf{I}-\mathbf{T}_H)] \end{array} \qquad (8.7c)$$

hence the space $(\mathbf{I}-\mathbf{T}_{HV})$ is a linear combination of $(\mathbf{I}-\mathbf{T}_H)$ and $(\mathbf{I}-\mathbf{T}_V)$. The conditions of symmetry imply that the linear weight vector $\mathbf{\Phi}$ *is orthogonal to the 3 dimensional sub-space of* R^4 defined by the two 2 dimensional spaces $(\mathbf{I}-\mathbf{T}_H)$ and $(\mathbf{I}-\mathbf{T}_V)$ overlapping in one dimension. Note however that the condition $\mathbf{\Phi}^T(\mathbf{I}-\mathbf{T}_{HV})=\mathbf{0}$ is still necessary in order to ensure that the combined application of $\mathbf{T}_H$, $\mathbf{T}_V$ abides to the conditions of symmetry.

Seeing that $\mathbf{\Phi}$ is orthogonal to a 3-D sub-space of R^4, there is an appropriate system of co-ordinates that makes $\mathbf{\Phi}$ 1-D. One can easily find a transform from equation (8.5) that identifies the suitable reduction

$$\mathbf{\Phi}^T(\mathbf{I}-\mathbf{T}_H)=[\phi_1\ \phi_2\ \phi_3\ \phi_4]\cdot\begin{bmatrix}1&-1&0&0\\-1&1&0&0\\0&0&1&-1\\0&0&-1&1\end{bmatrix}=\mathbf{0}\Rightarrow\begin{cases}\phi_1=\phi_2\\\phi_3=\phi_4\end{cases} \tag{8.8a}$$

$$\mathbf{\Phi}^T(\mathbf{I}-\mathbf{T}_V)=[\phi_1\ \phi_2\ \phi_3\ \phi_4]\cdot\begin{bmatrix}1&0&-1&0\\0&1&0&-1\\-1&0&1&0\\0&-1&0&1\end{bmatrix}=\mathbf{0}\Rightarrow\begin{cases}\phi_1=\phi_3\\\phi_2=\phi_4\end{cases} \tag{8.8b}$$

$$\mathbf{\Phi}^T(\mathbf{I}-\mathbf{T}_{HV})=[\phi_1\ \phi_2\ \phi_3\ \phi_4]\cdot\begin{bmatrix}1&0&0&-1\\0&1&-1&0\\0&-1&1&0\\-1&0&0&1\end{bmatrix}=\mathbf{0}\Rightarrow\begin{cases}\phi_1=\phi_4\\\phi_2=\phi_3\end{cases} \tag{8.8c}$$

Hence the linear equation is trasformed into:

$$[\phi_1\ \phi_2\ \phi_3\ \phi_4]\cdot[x_1\ x_2\ x_3\ x_4]^T=\phi_s(x_1+x_2+x_3+x_4) \tag{8.9}$$

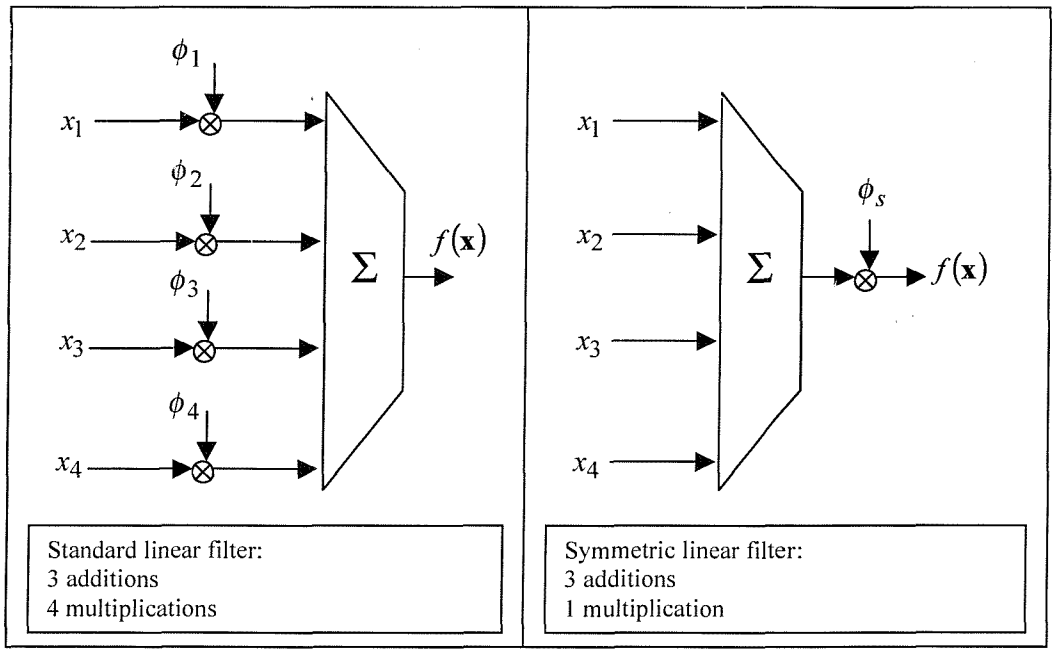

Figure 4. Architectural reduction for the square/rectangular lattice.

Equation (8.9) determines the architectural changes, and the computational savings obtained by making use of symmetry, as can be seen in figure 4. The same structure results for the rectangular lattice, and any symmetric lattice can be reduced in a similar way.

However, the results obtained so far have been deduced for two specific cases. It is possible to draw similar equations for any lattice which has symmetric properties, but it is desiderable to produce more general results. In order to achieve this the case of a continuous input space is considered. Although of little practical importance, the results obtained in the next section provide a greater theoretical breath to the reduction obtained so far and can be used to obtain a general criterion for the reduction of arbitrary symmetric lattices.

8.4. Symmetry in a continuous image space

In the continuous image space, an image is defined as a continuous function of two variables, $I(\mathbf{p}) = I(x,y)$. In the context of this discussion, the continuous linar interpolation of an image point $\mathbf{p}_0$ can be defined as the 2-D integral, over some neighbourhood $N_{\mathbf{p}_0}$ of $\mathbf{p}_0$, of the image function $I(\mathbf{p})$ multiplied by a *weight density function* $\Omega(\mathbf{p})$:

$$y(I) = \int_{N_{\mathbf{p}_0}} \Omega(\mathbf{p}) I(\mathbf{p})\, d\mathbf{p} \tag{8.10}$$

It is assumed that $\Omega(\mathbf{p})$ has closed support in R^2 (corresponding to the use of finite apertures in the discrete case), and assume very loose conditions on the shape of $N_{\mathbf{p}_0}$ to ensure that it is meaningful in terms of a real, discrete sampling lattice. It is also assumed for simplicity that the co-ordinate system is centred on $\mathbf{p}_0$.

8.4.1. Circular sampling and symmetry operators

In the following discussion a circular neighbourhood $N_{\mathbf{p}_0}$ of radius r and centered in $\mathbf{p}_0$ is considered. This choice is clearly justified in the context of the study of symmetric lattices. $I(\mathbf{p})$ can be rotated by any angle ϕ, and flipped around any axis (figure 5). Using polar co-ordinates centered at the origin then:

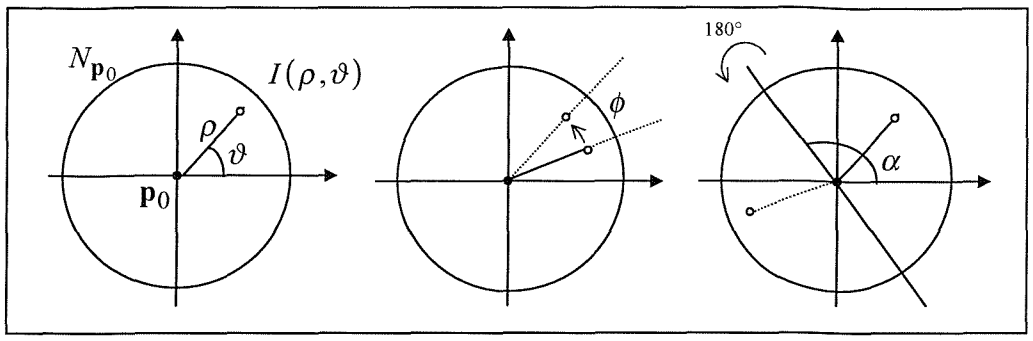

Figure 5. Circular sampling and symmetry operators.

$$y(I)=y(\rho,\vartheta)=\int_0^r\int_0^{2\pi}\Omega(\rho,\vartheta)\cdot I(\rho,\vartheta)d\vartheta d\rho \tag{8.11}$$

Where $I(\rho,\vartheta)$ is the image espressed in polar co-ordinates centred at $\mathbf{p}_0$. The next step is to find an explicit description of the operators $\mathbf{T}$ in the continuous space. The rotation operator $\mathbf{T}_{90}$ becomes a more general operator

$$\mathbf{T}_\phi: \quad \mathbf{T}_\phi\,[I(\rho,\vartheta)]=I(\rho,\vartheta-\phi) \tag{8.12a}$$

The operators $\mathbf{T}_H$, $\mathbf{T}_V$ and $\mathbf{T}_{HV}$ become the *mirror symmetry operator* about an axis α:

$$\mathbf{T}_\alpha: \quad \mathbf{T}_\alpha\,[I(\rho,\vartheta)]=I(\rho,2\alpha-\vartheta) \tag{8.12b}$$

The two operators are pictorially described in figure 5. These operators can be expressed as:

$$\mathbf{T}_\phi\,[I(\rho,\vartheta)]=\int_0^{2\pi}\delta(v-\vartheta+\phi)\cdot I(\rho,v)dv \tag{8.13a}$$

$$\mathbf{T}_\alpha [I(\rho,\vartheta)] = \int_0^{2\pi} \delta(v+\vartheta-2\alpha)\cdot I(\rho,v)dv \tag{8.13b}$$

8.4.2. Radial symmetry

The condition of symmetry in the continuous space can be formulated as follows:

$$y(I) = y\left(\mathbf{T}_\phi [I]\right),\ \forall\phi\in[0,2\pi],\ \forall I \tag{8.14a}$$

$$y(I) = y(\mathbf{T}_\alpha [I]),\ \forall\alpha\in[0,2\pi],\ \forall I \tag{8.14b}$$

which expresses rotational and mirror symmetry with arbitrary angles ϕ and α. It is assumed that (8.14a) and (8.14b) are valid for any image I. This leads to the following equations:

$$\int_0^r\int_0^{2\pi} \Omega(\rho,\vartheta)\{I(\rho,\vartheta)-\mathbf{T}_\phi [I(\rho,\vartheta)]\}d\vartheta\, d\rho = 0,\quad \forall\phi \tag{8.15a}$$

$$\int_0^r\int_0^{2\pi} \Omega(\rho,\vartheta)\{I(\rho,\vartheta)-\mathbf{T}_\alpha [I(\rho,\vartheta)]\}d\vartheta\, d\rho = 0,\quad \forall\alpha \tag{8.15b}$$

Substituting (8.13a) in (8.15a), we obtain:

$$\begin{aligned}&\int_0^r\int_0^{2\pi} \Omega(\rho,\vartheta)\cdot I(\rho,\vartheta)d\vartheta\, d\rho - \\ &\int_0^r\int_0^{2\pi} \Omega(\rho,\vartheta)\left[\int_0^{2\pi}\delta(v-\vartheta+\phi)I(\rho,v)dv\right]d\vartheta\, d\rho = 0,\quad \forall\phi\end{aligned} \tag{8.16}$$

By inverting the roles of v and ϑ in the second term of (8.16) and changing the order of integration, one obtains

$$\int_0^r \int_0^{2\pi} \Omega(\rho,\vartheta)\left[\int_0^{2\pi} \delta(v-\vartheta+\phi)I(\rho,v)dv\right]d\vartheta\, d\rho =$$

$$\int_0^r \int_0^{2\pi} I(\rho,v)\left[\int_0^{2\pi} \delta(v-\vartheta+\phi)\Omega(\rho,\vartheta)d\vartheta\right]dv\, d\rho = \qquad (8.17)$$

$$\int_0^r \int_0^{2\pi} I(\rho,v)\Omega(\rho,v+\phi)dv\, d\rho$$

By substituting (8.17) in (8.16), one can easily see that the conditions of symmetry for arbitrary angles imply that the *weight density function has radial symmetry.*

$$\Omega(\rho,\vartheta)=\Omega(\rho,\vartheta+\phi)=\Omega(\rho) \quad \forall\phi \qquad (8.18a)$$

and therefore,

$$y(I)=\int_0^r \int_0^{2\pi} \Omega(\rho,\vartheta)\cdot I(\rho,\vartheta)d\vartheta\, d\rho = \int_0^r \Omega(\rho)\int_0^{2\pi} I(\rho,\vartheta)d\vartheta\, d\rho \qquad (8.18b)$$

Note that (8.18b) justifies, in the continuous space, the reduction obtained for the discrete, square sampling case.

8.4.3. Mirror symmetry

In the scenario depicted previously, the image neighbourhood can be rotated by any arbitrary angle, and the interpolation must yield the same output. The consequence is that the weight density function has radial symmetry. However, it is difficult to translate this result in discrete terms, since sampling apertures usually have few axis

of symmetry (typically 2). The result (8.18a) has been obtained applying the radial symmetry operator (8.13a).

In this section we will focus our attention on the operator (8.13b), in the case of $\alpha = 0, \pi/2$ which matches 2 symmetries in the 8_1 sampling lattice (figure 6), and includes the 180° degree rotational symmetry. The missing lines in the discrete lattice (target field) can be modelled as missing zones (shaded areas) in the continuous neighbourhood.

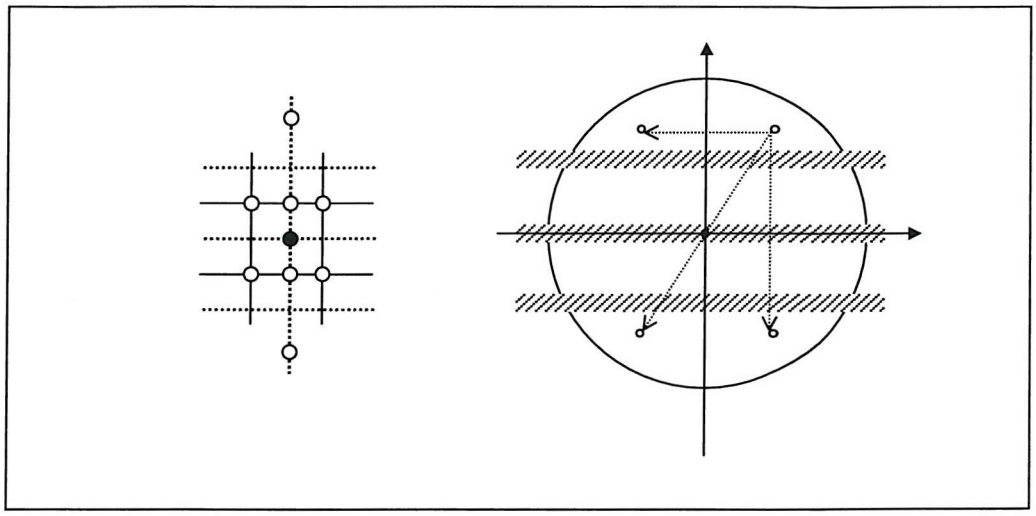

Figure 6. Aperture 8_1, and its continuous space model.

Following the derivation as in section 8.4.2 one can obtain an equation similar to (8.16), this time however using the specular symmetry operator (8.13b):

$$\int_0^r \int_0^{2\pi} \Omega(\rho,\vartheta) I(\rho,\vartheta) d\vartheta \, d\rho - \int_0^r \int_0^{2\pi} \Omega(\rho,\vartheta) \left[\int_0^{2\pi} \delta(\nu + \vartheta - 2\alpha) I(\rho,\nu) d\nu \right] d\vartheta \, d\rho = 0, \quad \alpha = 0, \frac{\pi}{2} \tag{8.19}$$

This time the equation is only valid for two values of α. Again, inverting the roles of ν and ϑ leads to:

$$\Omega(\rho,\vartheta)=\Omega(\rho,\pi-\vartheta)=\Omega(\rho,-\vartheta)=\Omega(\rho,-\pi+\vartheta) \tag{8.20}$$

In expressing (8.20) the fact that the axis of symmetry $\alpha=0,\pi/2$ are equivalent to $\alpha=\pi,-\pi/2$ has been used. The associated areas are schematically represented by the white dots in $N_{\mathbf{p}_0}$ in figure 6.

8.5. Symmetry in the discrete space

The results obtained in the previous section allow one to exploit the symmetric reductions of the linear interpolator for any lattice, without performing the analysis of the operators **T** seen in section 8.3. By simply inspecting the lattice, one can determine the available symmetries. The results in section 8.3 ensure that pixels associated by symmetry yield the same weight. Examples are shown in figure 7, where single weights values are associated with pixels shown with the same shading. The case of aperture 8_1 is included, as well as a more general lattice using all the available lines (i.e. without the missing lines typical of de-interlacing apertures). The latter example has an extra axis of symmetry (diagonal).

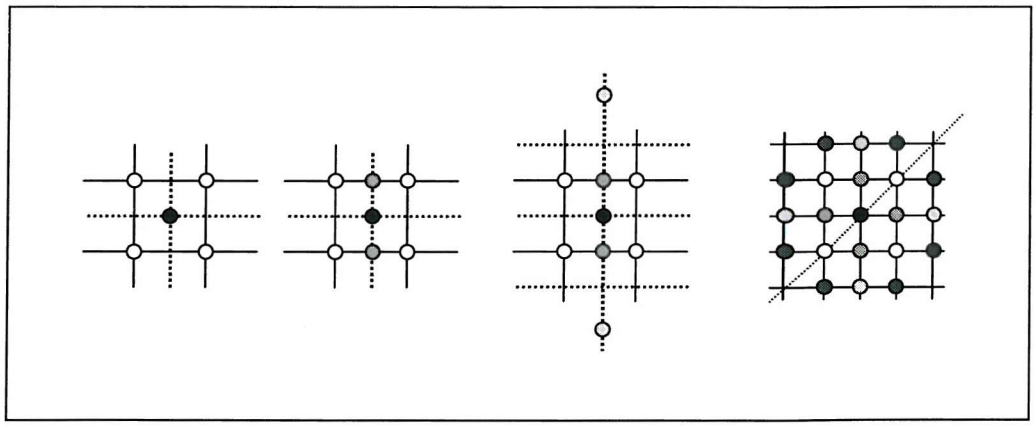

Figure 7. Symmetry in sampling lattices. Pixels with the same shading should be summed using the same weights.

The symmetric linear interpolator for the 8_1 aperture is given by

$$y(\mathbf{x})=\phi_1^{(s)}(x_1+x_8)+\phi_2^{(s)}(x_3+x_6)+\phi_3^{(s)}(x_2+x_4+x_5+x_7) \tag{8.21}$$

and therefore:

$$y(\mathbf{x}) = \mathbf{\Phi}^{(s)T}\mathbf{C}\mathbf{x} \tag{8.22}$$

$$\mathbf{C} = \begin{bmatrix} 1&0&0&0&0&0&0&1 \\ 0&0&1&0&0&1&0&0 \\ 0&1&0&1&1&0&1&0 \end{bmatrix} \quad \mathbf{\Phi}^{(s)} = \left[\phi_1^{(s)}\ \phi_2^{(s)}\ \phi_3^{(s)}\right]^T$$

The compression matrix $\mathbf{C}$ serves to reduce the dimension of the input from 8 to 3. The output of the linear de-interlacer is only a function of the 3-elements vector $\mathbf{C}\mathbf{x}$. Similar compression matrices can be defined for any discrete aperture.

8.5.1. Axial sampling.

In this section we want to show how the structure of the lattice is reflected in the flip operator $\mathbf{T}$ when some of the pixels lay on one of the axes of symmetry. For the 4 pixel square lattice depicted in figure 7, where all the pixels are off the axis, there is a reduction by a factor of 4 in the number of multipliers required to implement the scheme. For the 6 and 8 pixel apertures however, this reduction is 3 and 8/3 respectively. This is primarily because these apertures contain axial pixels.

Consider how the presence of axial pixels is reflected in the structure of the flip operators, and how this determines the reduction that can be obtained. For the 8_1 lattice the two operators are:

$$\mathbf{T}_V = \begin{bmatrix} 0&0&0&0&0&0&0&1 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 1&0&0&0&0&0&0&0 \end{bmatrix} \quad \mathbf{T}_H = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix} \tag{8.23}$$

The analysis for the 4-taps case can be repeated, as in section 8.3. However, here $\mathbf{T}_H$ has 4 entries on the diagonal, reflecting the fact that $\mathbf{T}_H$ maps axial taps into themselves (Figure 8). Hence:

$$\mathbf{T}_H - \mathbf{I} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.24}$$

There are 4 null columns, corresponding to the 4 axial pixels. Hence, $rank(\mathbf{T}_V - \mathbf{I}) = 4$, $rank(\mathbf{T}_H - \mathbf{I}) = 2$ and $rank(\mathbf{T}_H - \mathbf{I})^T(\mathbf{T}_V - \mathbf{I}) = 1$, so that the vector $\mathbf{\Phi}$ is orthogonal to a 5-dimensional space (4+2-1), and a reduction by 8/(8-5) = 8/3 is observed. The same reasoning can be used to explain the reduction by 3 obtained when the 6-pixel lattice in figure 6 is considered.

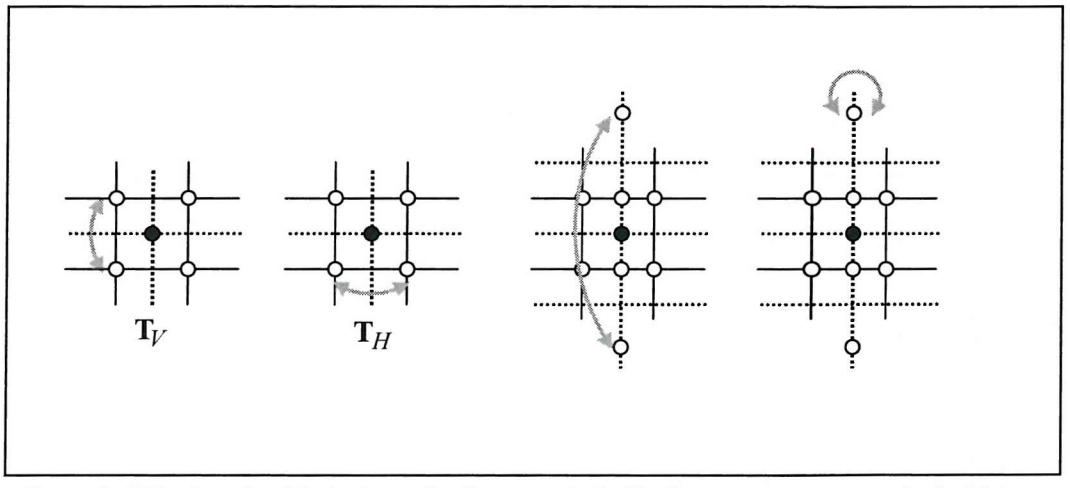

Figure 8. Off-axis and axial pixels. In the first case, both flipping operators map each pixel into a different pixel. In the second case, the horizontal flip maps axial pixels onto themselves. This is reflected as null columns in the orthogonal spaces $\mathbf{I}$-$\mathbf{T}_H$.

8.6. Experimental results.

These concepts are applied to the 3 symmetric sampling lattices depicted in figure 9. Of course, the 2-pixel lattice has only 1 axis of symmetry, hence one can only apply one flip operator, specifically $\mathbf{T}_V$, whilst the 4 pixel lattice and the 8-pixel lattice can be reduced with the two operators $\mathbf{T}_H$ and $\mathbf{T}_V$.

The experiments are conducted as follows: firstly, the MMSE filter is calculated, then, the image is "flipped" according to the flip operators the lattice permits, and the MSE

is computed when the original filter is applied to the flipped image. The results are compared with those obtained with the symmetric architecture, see table 1.

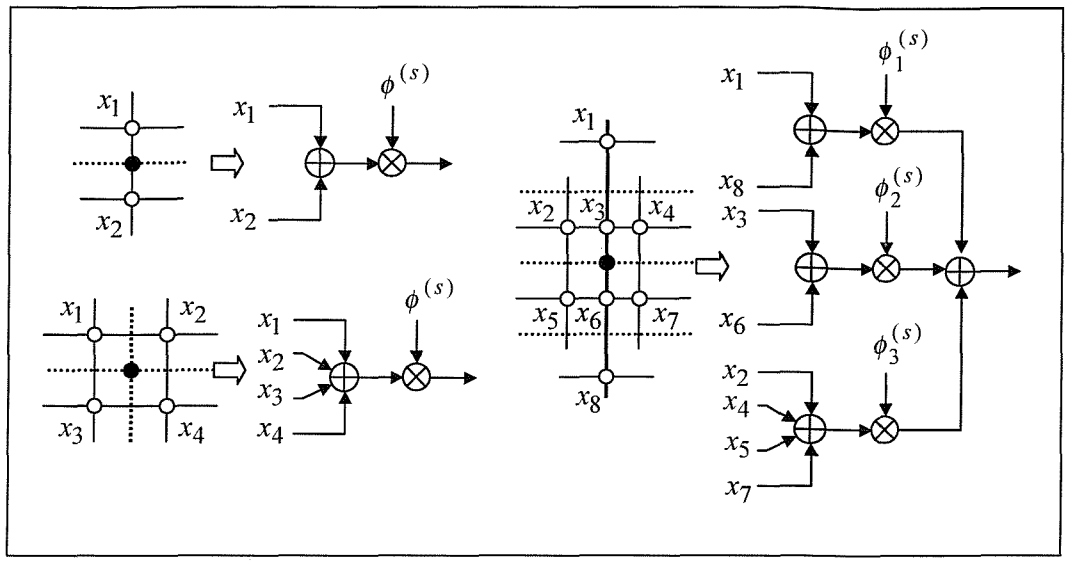

Figure 9. Sampling lattices, and symmetry reduced architectures.

	2 taps	4 taps	8 taps
Normal input	1.2369	2.7737	0.9893
H-flipped input	_____	2.9199	1.0596
V-flipped input	1.2379	2.9214	1.0616
HV-flipped image	_____	2.7752	0.9913
Symmetric architecture	1.2372	2.8106	1.0072

Table 1. Results for 3 sampling lattices, compared with symmetric architecture.

It can be seen that the non-symmetric architecture works better than the symmetric architecture on the original image, i.e. the training set. This can be interpreted as an excessive specialisation of the non-symmetric algorithm. The symmetric filter provides robust performance across the set of flipped images, so is more general. This can be regarded as a consequence of being more parsimonious. This benefit is gained along with computational savings.

8.7. Conclusions

It has been highlighted in previous sections how the computational demand of a de-interlacing system is of paramount importance. In single-layer networks, computational complexity is proportional to the number of nodes employed by the network, and to the complexity of the individual linear and non-linear kernels. Another advantage of seeking for smaller architectures is that this typically determine a more general result.

It is often possible to achieve computational savings by seeking for invariant features, e.g. invariance to translation and rotation. The first part of this chapter describes one of these invariant properties that can be postulated to be valid for any reasonable image. This invariance comes from the constancy of the target value under rotations of a symmetric sampling lattice. This determines a set of conditions (symmetry constraints) that a de-interlacing system should verify in order to preserve such rotational invariance. The case for a linear system is discussed in this chapter.

The conditions of symmetry for a linear system lead to simple linear algebraic equations. Initially, two very simple cases are presented, regarding a 4-tap sampling lattice with square and rectangular aperture. It is demonstrated that the conditions of symmetry imply that it is possible to reduce the dimensionality of the input without affecting the performance. This in turn allows a rearrangement of the algebra that leads to computational savings.

In order to generalise these results to arbitrary apertures, a brief discussion of the continuous case is presented. This discussion leads to the general principle that pixels associated by the symmetry operators should be multiplied by the same weight. This leads to a very simple procedure to determine the desired computational reduction.

9. SYMMETRY IN RADIAL BASIS FUNCTION NETWORKS

9.1. Introduction

The previous chapter highlighted the advantages of exploiting symmetry in linear interpolation and, specifically, in linear de-interlacing, and a theoretical foundation was presented for achieving this. In this chapter we seek to extend these concepts to a non-linear de-interlacing system based on RBFN.

For the linear case it has been shown how the output of a symmetric linear system depends on a reducing transformation of the input vector **x** expressed by

$$y(\mathbf{x}) = f(\mathbf{C}\,\mathbf{x}) \tag{9.1}$$

where **C** is the compression matrix that reduces the dimensionality of the input space. For the 8_1 aperture, **C** reduces the dimension of the input space from 8 to 3. It has been shown that, in the linear case, this reduction leads to computational savings.

A natural approach to applying the results of the last chapter to a non-linear system is to consider employing the same compression matrix. This is based on the fact that symmetry is a property of images that is valid regardless the particular technique adopted, and we can always write conditions of symmetry for the mapping function $f(\mathbf{x})$, provided the sampling lattice is symmetric.

Unfortunately the assumption of a linear interpolation is integral to the derivation of (9.1). A counter-example that illustrates this failing is to consider a 2 pixel non-linear de-interlacer given by $f(x_1, x_2) = \sqrt{x_1^2 + x_2^2}$. Evidently, this function satisfies the symmetry condition $f(x_1, x_2) = f(x_2, x_1)$ but $f(x_1, x_2)$ cannot be expressed in the form $f(\mathbf{C}\,\mathbf{x}) = f(x_1 + x_2)$.

A simple experiment shows the limitation of the linear approach in the case of RBFN. A 1000-centre, narrow support ($\gamma = 10^{-6} \sim 10^{-1}$) hybrid Gaussian RBFN is trained with the input space obtained using the 8_1 sampling lattice, and the input's dimension is reduced using the compression matrix **C**. Figure 1 depicts the MSE as the number of centres is increased and, despite using width parameters that differ by 5 orders of magnitude, the results are very similar and both are poor in comparison to the system using the full 8 dimensions.

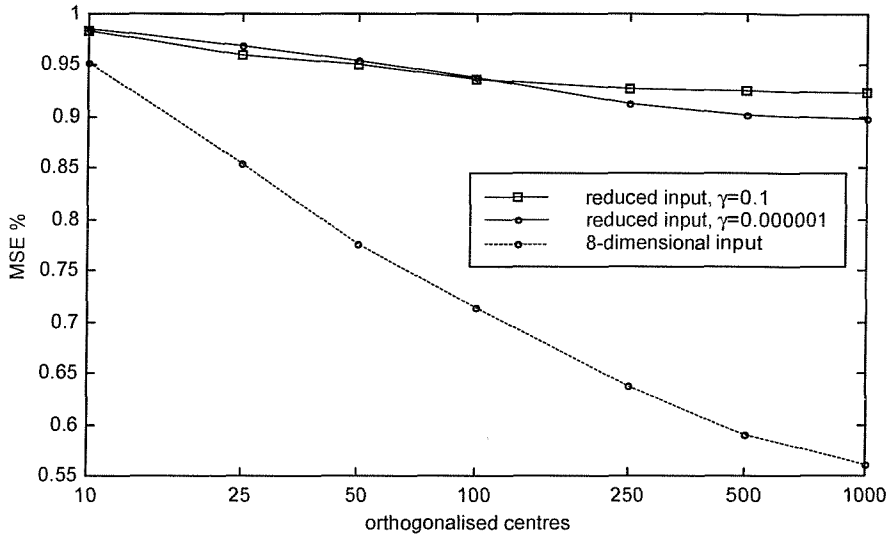

Figure 1. Gaussian RBFN trained with the reduced input set.

In this chapter two procedures to reduce RBFN complexity making use of symmetry will be presented. These procedures are based essentially on two different properties of the RBFN expansion.

Firstly, RBFN is a series of a non-linear $R^D \rightarrow R^M$ transform on the input space, followed by a linear regression $R^M \rightarrow R$ of the M dimensional intermediate space. The *symmetric RBFN model* described in section 9.2 is an attempt to transfer the symmetry properties of the input space to the intermediate space. This will result in a reduced vector $\mathbf{\Phi}$, that will result in a reduced number of output multiplications.

The second technique derives from considering the way the non-linear mapping is performed in RBFN. As a general principle, the distribution of the centres set $\mathbf{c}_i$, $i = 1 \ldots M$ tries to approximate the distribution of the input set $\mathbf{x}_n$, $n = 0 \ldots L-1$ in R^D. For $L = M$, we have that any input $\mathbf{x}_n$ has a corresponding centre $\mathbf{c}_i = \mathbf{x}_n$ (exact interpolation, section 3.3). In the general (and realistic) case, $L >> M$, one may assume that the centres set is distributed according to the input distribution. It seems reasonable to consider any input $\mathbf{x}_n$ as being "associated" with some centre $\mathbf{c}_i$, possibly its closest neighbour. This bond between centres and inputs, and the interpretation of symmetry as a relationship between different zones of the input space, will lead to the definition of *folding techniques* in section 9.3. Folding techniques lead to more general results than the symmetric model, since they use symmetry to reduce the complexity of the non-linear transformation. Furthermore, a folded RBFN model is also implicitly a symmetric model.

9.2. The symmetric RBFN model

This section considers one method by which symmetry can be enforced on a RBFN. The output of the RBFN is the outcome of a linear transform of the R^M intermediate space. In other words, the RBFN can be written as:

$$y(\mathbf{x}) = \sum_{i=1}^{M} \phi_i \, h(\|\mathbf{x} - \mathbf{c}_i\|; \sigma_i) = \sum_{i=1}^{M} \phi_i \, h_i(\mathbf{x}) = \mathbf{\Phi}^T \mathbf{h} \qquad (9.2)$$

where $\mathbf{h} = [h_1(\mathbf{x}) \ldots h_M(\mathbf{x})]^T$ is a vector in the intermediate space (spanning a sub-space of R^M), and $\mathbf{\Phi} = [\phi_1 \ldots \phi_M]^T$ is a set of weights that create the linear transform $R^M \rightarrow R$. Note that in (9.2) the dependence of $h_i(\cdot)$ on the parameter σ_i that controls the node's support in R^D is made explicit. For the 8_1 sampling lattice, the three flip operators, $\mathbf{T}_H$, $\mathbf{T}_V$ and $\mathbf{T}_{HV} = \mathbf{T}_H \cdot \mathbf{T}_V$ are those we seek invariance to.

9.2.1. Properties of flip operators and kernels

Consider the following definition: A RBFN is said to be symmetric to the operators $\mathbf{T}_H$, $\mathbf{T}_V$ and $\mathbf{T}_{HV}$ if

$\forall h_i = h(\|\mathbf{x}-\mathbf{c}_i\|;\sigma_i)$, $\exists h_i^{(H)}, h_i^{(V)} h_i^{(HV)}$ where

$$h_i^{(H)}(\mathbf{x}) = h(\|\mathbf{x}-\mathbf{T}_H\ \mathbf{c}_i\|;\sigma_i) \tag{9.3a}$$

$$h_i^{(V)}(\mathbf{x}) = h(\|\mathbf{x}-\mathbf{T}_V\ \mathbf{c}_i\|;\sigma_i) \tag{9.3b}$$

$$h_i^{(HV)}(\mathbf{x}) = h(\|\mathbf{x}-\mathbf{T}_{HV}\ \mathbf{c}_i\|;\sigma_i) \tag{9.3c}$$

In other words, the set of centres is closed to the flip operators, i.e. any flip of a centre results in a vector that is also a member of the centre set. From the properties of the general flip operator $\mathbf{T}$ (see chapter 8), it is easy to demonstrate that:

$$h_i^{(H)}(\mathbf{T}_H\ \mathbf{x}) = h_i(\mathbf{x}) \tag{9.4a}$$

$$h_i^{(H)}(\mathbf{T}_V\ \mathbf{x}) = h_i^{(HV)}(\mathbf{x}) \tag{9.4b}$$

$$h_i^{(H)}(\mathbf{T}_{HV}\ \mathbf{x}) = h_i^{(V)}(\mathbf{x}) \tag{9.4c}$$

$$h_i^{(V)}(\mathbf{T}_V\ \mathbf{x}) = h_i(\mathbf{x}) \tag{9.4d}$$

$$h_i^{(V)}(\mathbf{T}_H\ \mathbf{x}) = h_i^{(HV)}(\mathbf{x}) \tag{9.4e}$$

$$h_i^{(V)}(\mathbf{T}_{HV}\ \mathbf{x}) = h_i^{(H)}(\mathbf{x}) \tag{9.4f}$$

$$h_i^{(HV)}(\mathbf{T}_{HV}\ \mathbf{x}) = h_i(\mathbf{x}) \tag{9.4g}$$

$$h_i^{(HV)}(\mathbf{T}_H\ \mathbf{x}) = h_i^{(V)}(\mathbf{x}) \tag{9.4h}$$

$$h_i^{(HV)}(\mathbf{T}_V\ \mathbf{x}) = h_i^{(H)}(\mathbf{x}) \tag{9.4i}$$

Now, we group the 4 scalar kernels (the elements $h_i^{(\cdot)}$) into a single kernel vector:

$$\mathbf{h}_i^{(s)} = \left[h_i \;\; h_i^{(H)} \;\; h_i^{(V)} \;\; h_i^{(HV)}\right]^T \tag{9.5}$$

Under these conditions, equation 9.2 can be arranged as:

$$y(\mathbf{x}) = \sum_{i=1}^{M/4} \mathbf{\Phi}_i^{(s)T} \mathbf{h}_i^{(s)}(\mathbf{x}) \tag{9.6}$$

where $\mathbf{\Phi}_i^{(s)} = \left[\phi_i \;\; \phi_i^{(H)} \;\; \phi_i^{(V)} \;\; \phi_i^{(HV)}\right]^T$ is the weight vector for the i-th kernel vector.

9.2.2. Kernel vector flip operators

Given the properties (9.4a...i), it is easy to show that:

$$\mathbf{h}_i^{(s)}(\mathbf{T}_H \mathbf{x}) = \mathbf{T}_H^{(s)} \mathbf{h}_i(\mathbf{x}) \tag{9.7a}$$

$$\mathbf{h}_i^{(s)}(\mathbf{T}_V \mathbf{x}) = \mathbf{T}_V^{(s)} \mathbf{h}_i(\mathbf{x}) \tag{9.7b}$$

$$\mathbf{h}_i^{(s)}(\mathbf{T}_{HV} \mathbf{x}) = \mathbf{T}_{HV}^{(s)} \mathbf{h}_i(\mathbf{x}) = \mathbf{T}_H^{(s)} \mathbf{T}_V^{(s)} \mathbf{h}_i(\mathbf{x}) \tag{9.7c}$$

where

$$\mathbf{T}_H^{(s)} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad \mathbf{T}_V^{(s)} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{9.8}$$

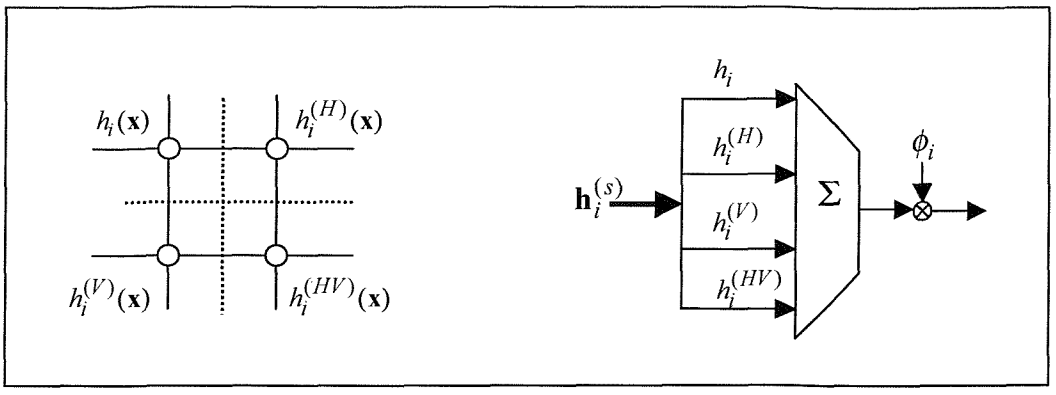

Figure 2. Left: formal sampling structure of the kernel vector. Right: reduction of the kernel vector.

The operators $\mathbf{T}_H^{(s)}$ and $\mathbf{T}_V^{(s)}$ have the formal structure of the flip operator defined on a 4-pixel rectangular lattice. In other words, the 8-dimensional flip of the input generates a 4-dimensional flip of the kernel vectors $\mathbf{h}_i^{(s)}$, $\forall i = 1 \ldots M/4$ (Figure 2). The two basic conditions of symmetry can be expressed as:

$$\sum_{i=1}^{M/4} \mathbf{\Phi}_i^{(s)T} \left[\mathbf{h}_i^{(s)}(\mathbf{x}) - \mathbf{h}_i^{(s)}(\mathbf{T}_H\, \mathbf{x}) \right] = \sum_{i=1}^{M/4} \mathbf{\Phi}_i^{(s)T} \left(\mathbf{I} - \mathbf{T}_H^{(s)} \right) \mathbf{h}_i^{(s)} = 0 \qquad (9.9a)$$

$$\sum_{i=1}^{M/4} \mathbf{\Phi}_i^{(s)T} \left(\mathbf{I} - \mathbf{T}_V^{(s)} \right) \mathbf{h}_i^{(s)} = 0 \qquad (9.9b)$$

From these identities it follows that:

$$\mathbf{\Phi}_i^{(s)T} \left(\mathbf{I} - \mathbf{T}_H^{(s)} \right) = 0 \qquad (9.10a)$$

$$\mathbf{\Phi}_i^{(s)T} \left(\mathbf{I} - \mathbf{T}_V^{(s)} \right) = 0 \qquad (9.10b)$$

and finally:

$$\mathbf{\Phi}_i^{(s)} = \phi_i [1\ 1\ 1\ 1]^T \qquad (9.11)$$

The result is that the flipping of the input vector can be related to flipping operators acting on the kernel vectors. It is interesting to note a difference between the input space and the intermediate space. The number of symmetries in the sampling lattice determines the number of flip operators acting on the input space. However, the rank of the matrices $\mathbf{T}$ is determined by the dimensionality of the input space. In the intermediate space the rank of $\mathbf{T}^{(s)}$ is equal to the number of symmetries in the lattice, and is therefore independent from the dimensionality of the input space.

9.2.3. Realisation of a symmetric RBFN

In a symmetric RBFN each kernel vector is simplified by the condition of symmetry, so that it requires 3 additions and 1 product. This results in a reduction by 4 of the number of output product units. The structure to achieve this is shown in figure 2. The symmetric model can be extended to arbitrary symmetric lattices, provided the kernel vector is created according to the number of flip operators defined by the symmetries of the lattice.

Consider the training of such a symmetric model. A simple technique is to take a large set of centres, $\mathbf{c}_i$, $i = 1 \ldots M$, then from each create 3 new centres: $\mathbf{T}_H\ \mathbf{c}_i$, $\mathbf{T}_V\ \mathbf{c}_i$ and $\mathbf{T}_{HV}\ \mathbf{c}_i$. This set of $4 \cdot M$ centres can then be orthogonalised, forcing the OLS to evaluate the combined effect of each set of 4 regressors, i.e. to save or discard the entire kernel vector $\mathbf{h}_i^{(s)}$ to preserve the symmetric structure. A downside of this approach is its inability to simplify the kernel vector. Suppose that $\mathbf{c}_i$ is such that $c_{i1} = c_{i8}, c_{i3} = c_{i6}$, and $c_{i2} = c_{i4} = c_{i5} = c_{i7}$ (assuming a 8_1 sampling lattice). In this case, we have $\mathbf{c}_i = \mathbf{T}_H\ \mathbf{c}_i = \mathbf{T}_V\ \mathbf{c}_i = \mathbf{T}_{HV}\ \mathbf{c}_i$. The kernel vector is redundant, and this redundancy will not be removed if we force OLS to consider the 4 kernels together. Figure 3 shows the results obtained using this training technique applied to a Gaussian RBFN.

Despite the degradation of the MSE performance, the proposed model offers a reasonable saving in the number of output multipliers (1/4). Note that the orthogonalisation in figure 3 has been performed on multiples of 4, in order to build the kernel vectors in the symmetric model.

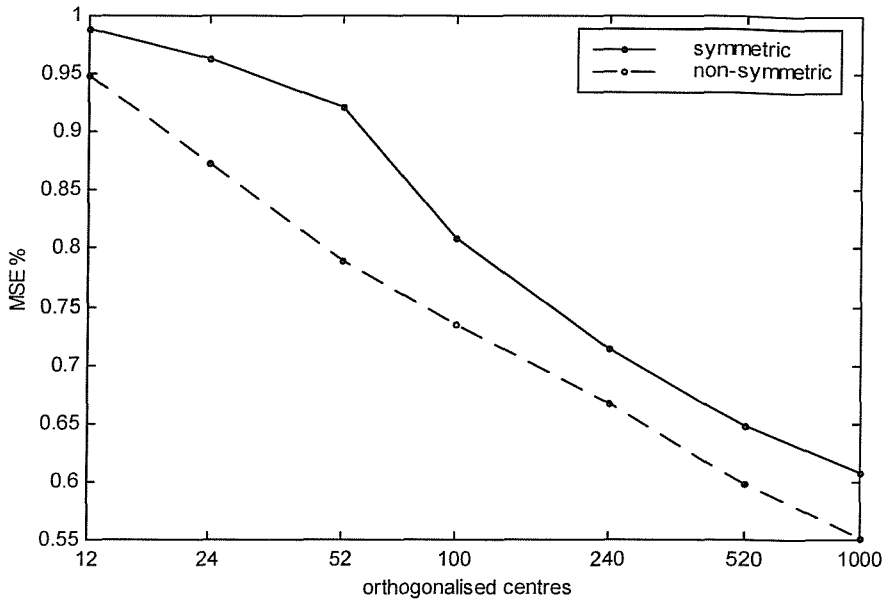

Figure 3. OLS training of the symmetric GRBFN. The horizontal axis shows the total number of scalar kernels (centres) involved.

The observed degradation maybe due to two factors. Firstly, the network is less specialised on the training set. This can be considered as a desirable effect, as it produces more general results. Firstly, the training procedure proposed is sub-optimal, since it does not remove redundant regressors in the kernel vector.

Unfortunately, the amount of computation involved in a RBFN is mainly dependent on M (the number of branches), i.e. on the dimensionality of the intermediate space, since most of the computational burden is associated with evaluating the $R^D \rightarrow R^M$ mapping (see appendix B). The symmetric RBFN, in which M is not reduced, only marginally reduces the computational load. Reducing M by means of symmetry involves the analysis of the non-linear core of the RBFN mapping.

9.3. Folding techniques in the input space

We have already underlined how RBFN are strongly related to a geometrical description of the input space and to the position centres occupy in this space. In simple words, centres map the input distribution, and hence work as prototypes of the input set. The training phase aims to find the set of output weights that minimises the

MSE, i.e. such that these prototypes generate the best approximation of the output dynamic (target sequence) associated with the input.

In this section a novel technique is presented that reduces the number of nodes in a RBFN trained to de-interlace a frame using a symmetric sampling lattice. This reduction is achieved by reducing the span of the input set in the input space, therefore reducing the number of centres required to yield the desired mapping. The technique is based on the principle that, since two input vectors associated by a flip operator $\mathbf{T}$ yield the same output, it is possible to merge them into a single input vector without affecting the performance. This principle is called folding, and the techniques developed in the next sections are called folding techniques.

9.3.1. The folding principle

The folding principle can be described as follows. The condition of symmetry in its most general form states that, given the input $\mathbf{x}$ and the flip transform $\mathbf{T}$, then $f(\mathbf{x}) = f(\mathbf{T}\mathbf{x})$. The input $\mathbf{x}$ and its flipped version $\mathbf{T}\mathbf{x}$ generally lie in different positions of the input space. Suppose that our interpolation system is able to discriminate the two inputs, and if presented with $\mathbf{T}\mathbf{x}$ as an input, moves it to the position occupied by $\mathbf{x}$. This movement is easily obtained applying the operator $\mathbf{T}$, since $\mathbf{T}(\mathbf{T}\mathbf{x}) = \mathbf{x}$. Given the condition of symmetry, such a movement will not affect the performance of the interpolator.

Now, suppose that, in an RBFN-based system, the two inputs are most closely associated with the two centres $\mathbf{c}$ and $\mathbf{T}\mathbf{c}$. Since the position formerly occupied by $\mathbf{T}\mathbf{x}$ is now empty, it seems reasonable to remove the $\mathbf{T}\mathbf{c}$ term, or alternatively to move $\mathbf{T}\mathbf{c}$ into the position of $\mathbf{c}$, since $\mathbf{c}$ will now represent the moved input $\mathbf{T}\mathbf{x}$ as well as $\mathbf{x}$. This concept is graphically represented in figure 4, in the two dimensional case. Note that the 2-dimensional span of the input set is reduced by half. In other words, the discrimination and movement system makes the input distribution more compact, at the same time yielding a similar output. Since the input space is compressed by a factor of 2 one might anticipate a corresponding reduction in the number of centres needed. In the 2 dimensional example, it is easy to find a suitable

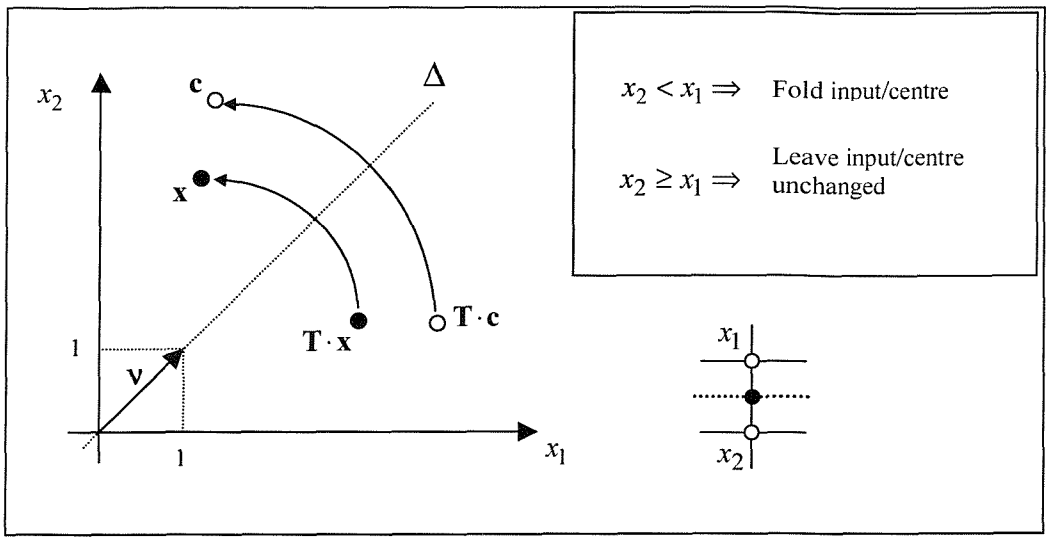

Figure 4. Folding in the 2 dimensional input space, and folding algorithm.

algorithm that halves the area of the input space. The algorithm is described in the inset in figure 4.

Basically, the input space is folded along the quadrant bisector Δ. If $D > 2$ this graphical representation fails, and one has to determine suitable alternatives that transcend the visual approach.

The folding principle has a downside, at least when applied to RBFN. Since the span of the input set is halved, we expect a similar MSE with half the number of centres. However the kernels $h(\|\mathbf{x}-\mathbf{c}\|;\sigma)$ usually have infinite support in R^D. Hence, the kernel generated by the centre $\mathbf{Tc}$, that represents the removed (folded) input $\mathbf{Tx}$, will have an influence on the interpolation function at $\mathbf{x}$. This problem is graphically represented in the left frame of figure 5. One may assume that centres far from the bisector Δ have little influence in the opposite half-quadrant, if the parameter σ is reasonably small. On the other hand, if $\mathbf{Tc}$ is very close to Δ we may assume that it significantly overlaps its folded version $\mathbf{T}(\mathbf{Tc})$, and hence is largely redundant. These limitations are depicted in the right frame of figure 5.

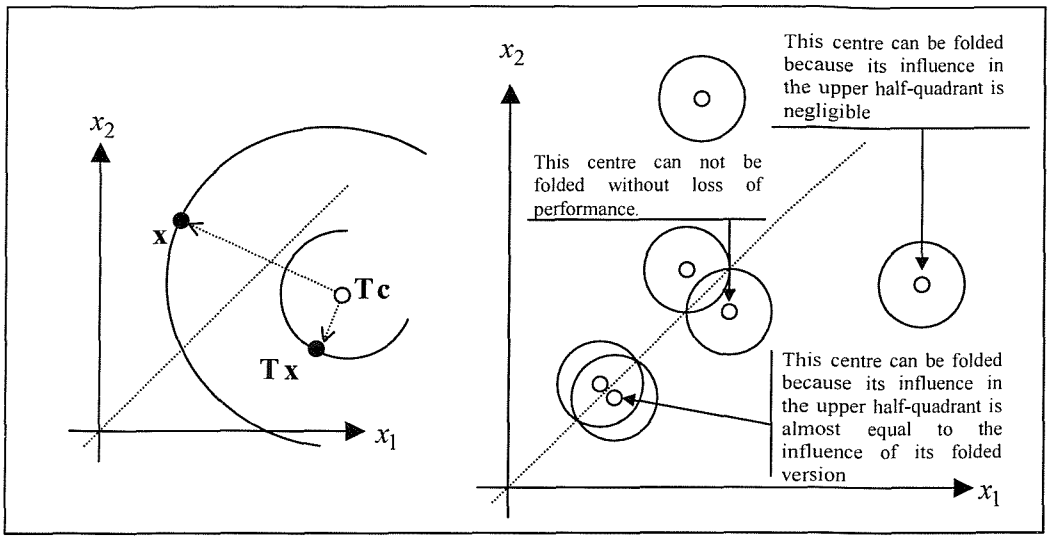

Figure 5. Effects of kernel's support on folding.

A folding strategy based on the geometrical properties of the flip operator in the plane has been devised in R^2. The flip operator mirrors inputs and centres about the bisector Δ, therefore the folding strategy is based on the position of the input/centre in the quadrant with respect to Δ.

Unfortunately, with $D>2$ such strategy is not achievable in the same way, for the reason that, although it is possible to consider the main diagonal Δ in R^D as the natural equivalent of the bisector in R^2, it is not possible to unambiguously determine whether a vector is "above" or "below" Δ. Furthermore, with lattices that extend in both the horizontal and vertical directions, more than one flip operator is usually required. Consequently, more than one decision has to be made.

However, the principle that different inputs associated by the flip operators can be folded onto each other is a concept independent of the dimension D of the input space and of the number of operators involved. In the following sections methods that extend this principle to arbitrary dimensions are discussed.

To achieve this goal, a proper geometrical abstraction of the folding principle is needed. Initially the general topological properties of the generic flip operator in R^D are discussed.

9.3.2. Topological properties of the flip operator in R^D

From figure 4, one can identify a series of properties that $\mathbf{T}$ satisfies in R^2. Firstly, the bisector Δ defines the set of fixed points of $\mathbf{T}$, i.e. a vector in Δ is folded onto itself. In R^D the bisector Δ is more properly described as the main diagonal of the input space. Δ can be mathematically described as the span of the vector $\boldsymbol{\nu} = \nu[1 \cdots 1]^T$, with $\nu \in [-128, 127]$. The span of $\boldsymbol{\nu}$ defines the set of inputs whose elements are all equal, and $\boldsymbol{\nu}$ satisfies:

$$\mathbf{T}(\alpha\boldsymbol{\nu}) = \alpha\boldsymbol{\nu}, \ \forall \mathbf{T}, \alpha \tag{9.12}$$

Differently from the two-dimensional case, in R^D Δ is only a subset of the set of fixed points of $\mathbf{T}$. Given an arbitrary $\mathbf{T}$, the set of its fixed points is properly defined as the sub-space $F_{\mathbf{T}} \subset R^D$ such that $\mathbf{x} \in F_{\mathbf{T}} \rightarrow \mathbf{T}\mathbf{x} = \mathbf{x}$. Hence $\mathbf{x} \in F_{\mathbf{T}}$ must satisfy the equation:

$$\mathbf{x} \in F_{\mathbf{T}} \rightarrow (\mathbf{T} - \mathbf{I})\mathbf{x} = \mathbf{0} \tag{9.13}$$

From (9.13) it follows that the space of fixed-points abides by the condition:

$$F_{\mathbf{T}}^{T}(\mathbf{T} - \mathbf{I}) = \mathbf{0} \tag{9.14}$$

Equation (9.14) defines a sub-space of R^D, depending on the particular operator considered. In many cases $dim(F_{\mathbf{T}}) > 1$, and hence $dim(F_{\mathbf{T}}) > dim(\Delta)$. However, given a general set of operators $\mathbf{T}_i$, the corresponding fixed-point spaces will all intersect at Δ. As an example, consider the R^4 case corresponding to the rectangular

lattice described in section 8, and its horizontal flip operator $\mathbf{T}_H$. It can be shown that $rank(\mathbf{I}-\mathbf{T}_H)=2$, hence $dim(F_{\mathbf{T}_H})=2$ and $F_{\mathbf{T}_H}$ is a 2 dimensional subspace of R^4. Further it can be shown that $dim(F_{\mathbf{T}_V})=2$, and $F_{\mathbf{T}_V} \cap F_{\mathbf{T}_H} = \Delta$.

There are two other interesting properties of $\mathbf{T}$ in R^2. Firstly, the segment $\mathbf{x}-\mathbf{T}\mathbf{x}$ is perpendicular to Δ. Secondly, $\mathbf{x}$ and $\mathbf{T}\mathbf{x}$ are equidistant from Δ. These properties extend to the general case in R^D for an arbitrary $\mathbf{T}$. The general properties of $\mathbf{T}$ noted in section 8 will be used to prove these results. Specifically, that $\mathbf{T}$ is equal to its inverse, i.e. $\mathbf{T}$ is orthogonal, which implies that $det(\mathbf{T})=1$.

Consider the projection (scalar product) of the vector difference $\mathbf{x}-\mathbf{T}\mathbf{x}$ over the vector $\mathbf{v}$. Using (9.12),

$$\mathbf{v}^T(\mathbf{x}-\mathbf{T}\mathbf{x})=\mathbf{v}^T(\mathbf{I}-\mathbf{T})\mathbf{x} \tag{9.15a}$$

but since $\mathbf{v}^T=\mathbf{v}^T\mathbf{T}$ we have:

$$\mathbf{v}^T(\mathbf{x}-\mathbf{T}\mathbf{x})=0 \tag{9.15b}$$

Hence $\mathbf{x}-\mathbf{T}\mathbf{x}$ is orthogonal to Δ. The distance between $\mathbf{T}\mathbf{x}$ and Δ is given by:

$$dist(\mathbf{T}\mathbf{x},\Delta)=\left\| \mathbf{T}\mathbf{x}-\left(\mathbf{v}^T\mathbf{T}\mathbf{x}\right) \right\| \tag{9.16a}$$

but from the isometric property of $\mathbf{T}$ it follows that:

$$dist(\mathbf{T}\mathbf{x},\Delta)=\left\| \mathbf{x}-\left(\mathbf{v}^T\mathbf{x}\right) \right\| = dist(\mathbf{x},\Delta) \tag{9.16b}$$

Hence $\mathbf{T}\mathbf{x}$ and $\mathbf{x}$ are equidistant from Δ. Note that the kernel vector in the symmetric model (section 9.2) has a well-defined topological structure that benefits from the

analysis conducted in this section. The important point is that a methodology of abstraction has been defined, that will be used in the subsequent sections to determine a possible folding algorithm in R^D.

9.3.3. Analysis of the eigenspace of the flip operators

This section describes another important framework of analysis: the general properties of the eigenvalues and eigenvectors sets of the folding operator. Before proceeding further, recall the property of $\mathbf{T}$ being a full-rank matrix, with $rank(\mathbf{T})=D$ and $det(\mathbf{T})=1$. This property arises from the general definition of $\mathbf{T}$ as an isometric rearrangement of the co-ordinate system in R^D. In other words, $\mathbf{T}$ is a "shuffling" without duplication of the columns of the identity matrix in R^D.

Alternatively, $\mathbf{T}$ can be described as an operator that associates each pixel with one and only one other pixel. A pixel can also be associated with itself, for example x_1 and x_8 in the lattice 8_1 under the action of the horizontal operator $\mathbf{T}_H$. This property leads to a straightforward method to calculate the D eigenvectors of $\mathbf{T}$. Suppose that the flip operator $\mathbf{T}$ associates the pixels x_i and x_j, and define the vector

$$\mathbf{v}^{(I)} = \frac{1}{\sqrt{2}}\left[0\cdots \underset{i-1}{0}\ \underset{i}{1}\ \underset{i+1}{0}\ \cdots\ \underset{j-1}{0}\ \underset{j}{1}\ \underset{j+1}{0}\ \cdots 0\right]^T \tag{9.17}$$

with null entries except for the i-th and the j-th element, and with $\| \mathbf{v}^{(I)} \| = 1$. It is clear that $\mathbf{T}\mathbf{v}^{(I)} = \mathbf{v}^{(I)}$. In the same way, given the vector

$$\mathbf{v}^{(V)} = \frac{1}{\sqrt{2}}\left[0\cdots \underset{i-1}{0}\ \underset{i}{1}\ \underset{i+1}{0}\ \cdots\ \underset{j-1}{0}\ \underset{j}{-1}\ \underset{j+1}{0}\ \cdots 0\right]^T \tag{9.18}$$

we have $\mathbf{T}\mathbf{v}^{(V)} = -\mathbf{v}^{(V)}$. Clearly, $\mathbf{v}^{(I)}$ and $\mathbf{v}^{(V)}$ are both eigenvectors of $\mathbf{T}$, with eigenvalues respectively +1 and –1. Since $\mathbf{v}^{(I)}$ is left unchanged by $\mathbf{T}$, both in norm and direction, we refer to it as an *invariant* eigenvector. Conversely $\mathbf{v}^{(V)}$ is termed a *variant* eigenvector.

Now, suppose that $\mathbf{T}$ associates the pixel x_i with itself. Then, the vector

$$\mathbf{v}^{(I)} = \begin{bmatrix} 0 \cdots & \underset{i-1}{0} & \underset{i}{1} & \underset{i+1}{0} & \cdots 0 \end{bmatrix}^T \tag{9.19}$$

is clearly an invariant eigenvector of $\mathbf{T}$. One can generalise these results stating that for each pixel association, that is not a self-associated, the eigenspace contains one invariant eigenvector and one variant eigenvector. On the other hand, if $\mathbf{T}$ associates a pixel with itself, then its eigenspace contains the appropriate invariant eigenvector. The eigenspace of $\mathbf{T}$ is completely described in terms of invariant and variant eigenvectors. Since $rank(\mathbf{T}) = D$, the set of all its eigenvectors defines a basis for R^D. Hence it is possible to divide R^D in an invariant space defined by the span of $\mathbf{V}^{(I)} = \bigcup \mathbf{v}^{(I)}$ and into a variant space defined by the span of $\mathbf{V}^{(V)} = \bigcup \mathbf{v}^{(V)}$.

The input $\mathbf{x}$ can be projected into these eigenspaces. Therefore, $\mathbf{x}$ can be divided into an invariant part $\mathbf{x}^{(I)}$ and a variant part $\mathbf{x}^{(V)}$, $\mathbf{x} = \left[\mathbf{x}^{(I)T} \ \mathbf{x}^{(V)T}\right]^T$. Clearly,

$$\mathbf{T}\left[\mathbf{x}^{(I)T} \ \mathbf{x}^{(V)T}\right]^T = \left[\mathbf{x}^{(I)T} \ -\mathbf{x}^{(V)T}\right]^T \tag{9.20}$$

If $\hat{f}\left(\mathbf{x}^{(I)}, \mathbf{x}^{(V)}\right)$ is the expression of $f(\mathbf{x})$ in the eigenspace, we can express the condition of symmetry for the operator $\mathbf{T}$ as:

$$\hat{f}\left(\mathbf{x}^{(I)}, \mathbf{x}^{(V)}\right) = \hat{f}\left(\mathbf{x}^{(I)}, -\mathbf{x}^{(V)}\right) \tag{9.21}$$

implying that $\hat{f}$ is even in the variant space. Finally, it can be shown from (9.21) that the locus of the fixed points of $\mathbf{T}$ is equal to the invariant space $\mathbf{V}^{(I)}$.

9.3.4. Binary classification of inputs and cluster reduction

This section discusses the extension of the decision algorithm to D dimensions. As already discussed, the problem in high-dimensional spaces is not only the ambiguity of the concept "above" and "below" Δ, but also the presence of more than one flip operator.

One method to extend the decision algorithm is as follows. Consider the 2-dimensional case first. Given the folding operator $\mathbf{T}$, define a *binary class function* $\kappa_{\mathbf{T}}(\mathbf{x}): \mathbf{X} \subset R^2 \rightarrow \{0,1\}$. Specifically, $\kappa_{\mathbf{T}}(\mathbf{x})=1$ if $x_1 \geq x_2$, $\kappa_{\mathbf{T}}(\mathbf{x})=0$ otherwise. In other words, $\kappa_{\mathbf{T}}(\mathbf{x})=0$ if $\mathbf{x}$ is in the lower half-quadrant, and hence $\mathbf{x}$ is said to belong to class 0. On the other hand, if $\mathbf{x}$ is in the upper half-quadrant (including Δ), then $\kappa_{\mathbf{T}}(\mathbf{x})=1$, and consequently $\mathbf{x}$ is said to belong to class 1. With these definitions, the folding algorithm in R^2 simply determines the class of the input, and decides to leave the input unchanged if $\kappa_{\mathbf{T}}(\mathbf{x})=1$ or to fold it in the upper half-quadrant if $\kappa_{\mathbf{T}}(\mathbf{x})=0$. Note that by definition $\kappa_{\mathbf{T}}(\alpha \mathbf{v})=1, \forall \alpha$. This choice is arbitrary and one may equally assign Δ to class 0. However this choice would be computationally more expensive, because the flip operator would be applied to the set of fixed points.

There are several characteristics of $\kappa_{\mathbf{T}}(\mathbf{x})$ in R^2 that we wish to focus on. The first is that the flip operator in R^2 corresponds to the algebraic NOT operation in the Boolean output space of the binary class function. In other words,

$$\kappa_{\mathbf{T}}(\mathbf{x}) = \overline{\kappa_{\mathbf{T}}(\mathbf{T}\mathbf{x})} \tag{9.17}$$

It is trivial to show that the two sets $\mathrm{K}_1 = \{\mathbf{x}: \kappa_{\mathbf{T}}(\mathbf{x})=1\}$ and $\mathrm{K}_0 = \{\mathbf{x}: \kappa_{\mathbf{T}}(\mathbf{x})=0\}$ are such that $\mathrm{K}_0 \cup \mathrm{K}_1 = \mathbf{X}$ and $\mathrm{K}_0 \cap \mathrm{K}_1 = \varnothing$ (the empty set). More importantly, $\mathrm{K}_0 \xleftrightarrow{\mathbf{T}} \mathrm{K}_1$, if $\mathbf{x} \notin F_{\mathbf{T}}$. Formally the operator $\mathbf{T}$ *is closed* in the set $\mathrm{K}_0 \cup \mathrm{K}_1$. The set $\Gamma = \mathrm{K}_0 \cup \mathrm{K}_1$ is called a *cluster*. In R^2 there is only one cluster, corresponding to the whole input set. As we will see, in higher-dimensional spaces and with more than

one operator there will be more than one cluster. Note that $\kappa_{\mathbf{T}}(\mathbf{x})$ operates on the variant projection of the input, $\mathbf{x}^{(V)}$, that in R^2 is equal to $(x_1 - x_2)$, so that one can write $\kappa_{\mathbf{T}}(\mathbf{x}) = \kappa_{\mathbf{T}}(\mathbf{x}^{(V)})$.

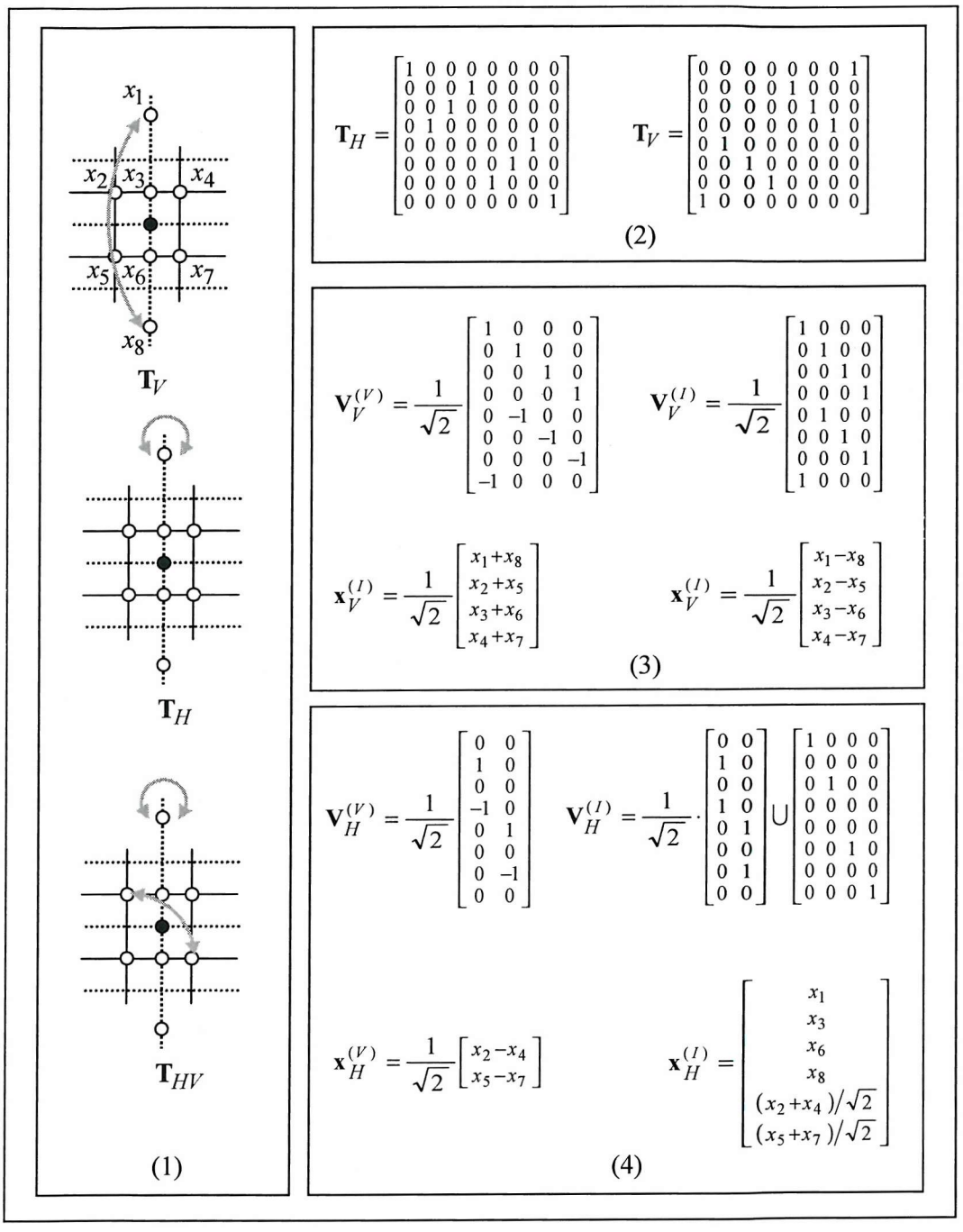

Figure 6. Properties of the flip operators for the lattice 8_1 depicted in (1). Matrix form of the independent operators (2). Variant and invariant eigenspaces, variant and invariant inputs for the vertical flip operator (3). Variant and invariant eigenspaces, variant and invariant inputs for the horizontal flip operator (4).

These concepts form the basis for our abstraction that can be summarised in the following general properties of the folding algorithm:

- The folding algorithm is based on an operator closed in the cluster.
- The algorithm folds one of the classes in the cluster onto the other class, halving the number of classes in each cluster.
- The decision is made according to a binary function (binary class function).
- The particular flip operator considered determines clusters and relative classes, the decision strategy and the binary class function.
- The binary class function operates on the variant space defined by the operator considered.

These properties can be used to describe a possible folding strategy in the input space R^8 defined by the 8_1 sampling lattice.

9.3.5. Binary classification of the 8_1 sampling lattice

For the 8_1 sampling lattice two flip operators can be defined, specifically the horizontal flip operator $\mathbf{T}_H$ and the vertical flip operator $\mathbf{T}_V$. It is also possible to define an horizontal-vertical operator $\mathbf{T}_{HV} = \mathbf{T}_H\ \mathbf{T}_V$. The flip performed on the lattice by these operators, their explicit matrix form and their invariant and variant space basis, together with the explicit expression of the invariant and variant projections of the input $\mathbf{x}$ are summarised in figure 6. Note how $\mathbf{T}_H$ generates an asymmetric set of eigenvalues, since the taps x_1, x_3, x_6 and x_8 flip onto themselves. We now consider a folding algorithm coherently with the general principles defined in the previous sections.

Since the variant spaces are 2 and 4 dimensional, the binary class functions define a 2 and 4 bit resolution output respectively. This leads to 16 classes and 8 clusters for $\mathbf{T}_V$, and 4 classes and 2 clusters for $\mathbf{T}_H$, as summarised in tables 1 and 2.

CLASS	$(x_1 - x_8) > 0$	$(x_2 - x_5) > 0$	$(x_3 - x_6) > 0$	$(x_4 - x_7) > 0$	CLUSTER
0	0	0	0	0	0
15	1	1	1	1	
1	0	0	0	1	1
14	1	1	1	0	
2	0	0	1	0	2
13	1	1	0	1	
3	0	0	1	1	3
12	1	1	0	0	
4	0	1	0	0	4
11	1	0	1	1	
5	0	1	0	1	5
10	1	0	1	0	
6	0	1	1	0	6
9	1	0	0	1	
7	0	1	1	1	7
8	1	0	0	0	

Table 1. Classes and clusters for the operator T_V. Note that we numbered the classes according to the base-10 value of their binary class function.

CLASS	$(x_2 - x_4) > 0$	$(x_5 - x_7) > 0$	CLUSTER
0	0	0	0
3	1	1	
1	0	1	1
2	1	0	

Table 2. Classes and clusters for the operator T_H

The two sets of clusters defined by $\mathbf{T}_H$ and $\mathbf{T}_V$ are independent, in the sense that the class of $\mathbf{x}$ given by the first operator's binary class function does not determine the class given by the second operator. Hence, two independent decision strategies for $\mathbf{T}_H$ and $\mathbf{T}_V$ can be specified, hence including a decision strategy for $\mathbf{T}_{HV}$.

For each operator the classes defined in tables 1 and 2 can be grouped into two groups according to the value of their most significant bit MSB. Since $\kappa_{\mathbf{T}}(\mathbf{x}) = \overline{\kappa_{\mathbf{T}}(\mathbf{T} \cdot \mathbf{x})}$, the folding operator always moves the input from one group to another. Note that the

MSB in $\kappa_{\mathbf{T}}(\mathbf{x})$ is determined by the order in which the pixels are sorted. From the beginning of this work, pixel numbering has reflected the natural line-scan order in video broadcast systems, but this choice is arbitrary. Any of the bits in $\kappa_{\mathbf{T}}(\mathbf{x})$ could correspond to the MSB by a simple rearrangement of the sampling lattice. This leads at least to 4 possible algorithms:

Apply $\mathbf{T}_V$ if $(x_1 - x_8) > 0$, and $\mathbf{T}_H$ if $(x_2 - x_4) > 0$

Apply $\mathbf{T}_V$ if $(x_3 - x_6) > 0$, and $\mathbf{T}_H$ if $(x_2 - x_4) > 0$

Apply $\mathbf{T}_V$ if $(x_1 - x_8) > 0$, and $\mathbf{T}_H$ if $(x_5 - x_7) > 0$

Apply $\mathbf{T}_V$ if $(x_3 - x_6) > 0$, and $\mathbf{T}_H$ if $(x_5 - x_7) > 0$

In all 4 cases, the decision problem in R^8 is reduced to a decision in two distinct R^2 spaces. This has an immediate consequence in the way the input space is folded.

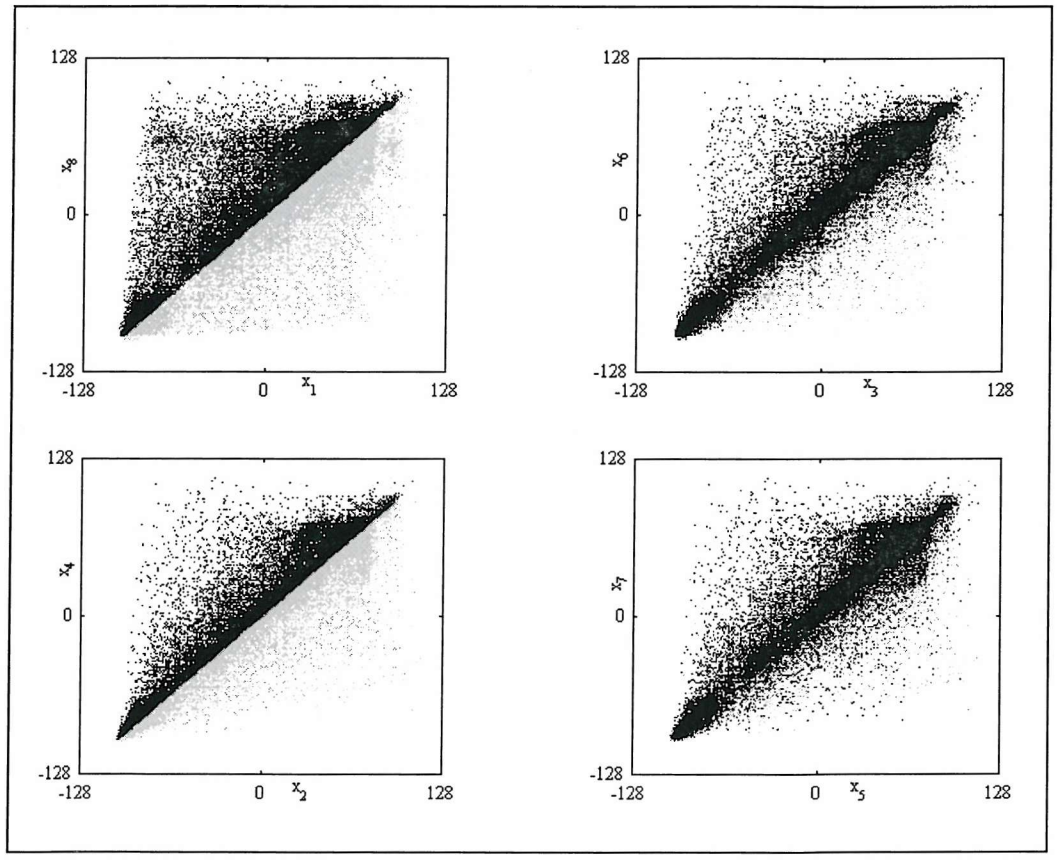

Figure 7. Folded Input space, algorithm 1. Original input space (grey), folded input space (black).

Figure 7 shows the projected planes of the input space folded using algorithm 1. Note the sharp cut in plane $x_1 - x_8$ and plane $x_2 - x_4$ (the planes where the decision algorithm is applied). However, there is also some reduction in the planes $x_3 - x_6$ and $x_5 - x_7$. This can be explained by considering that if $x_1 - x_8 > 0$, there is a reasonable probability of a local vertical gradient, so one might anticipate that also $x_3 - x_6 > 0$. The same consequence can be deduced in presence of a horizontal gradient, and hence for the plane $x_5 - x_7$.

After being used to fold the input set, the chosen algorithm must be applied to the centres of a Gaussian RBFN to achieve the desired reduction. However, as already introduced in section 9.3, there is the problem of the influence of each class of centres on the other classes. Since the choice of algorithms determine a particular "movement" of the folded centres in the input space, the different algorithms will, in general, yield different results. The issue of determining the best algorithm among the 4 possible choices should be addressed in future work.

Note that the cluster folding technique embeds the symmetric model described in section 9.2. In fact, the folding algorithm reduces the kernel vector $\mathbf{h}_i^{(s)}(\mathbf{x})$ to one of its scalar components, consequently reducing the number of output multipliers by 4.

9.3.6. Results for binary cluster folding RBFN

Figure 8 shows the OLS reduction of the four proposed folding strategies applied to HGRBFNs. The results are compared with the OLS reduction of a standard (unfolded) network. Firstly note that the four results are very similar. For networks with $M = 50 \sim 200$ one can approximately halve the number of centres and retain a similar MSE as the unfolded network, regardless the folding strategy chosen. For M>200, it seems that cluster folding looses its effectiveness, and the same may be true for M<50. Secondly, since the volume of the input set in R^8 has been reduced by 4, one may anticipate an equivalent reduction in the number of centres. It is somewhat surprising to observe only a halving of the number of centres necessary to achieve a given level of performance.

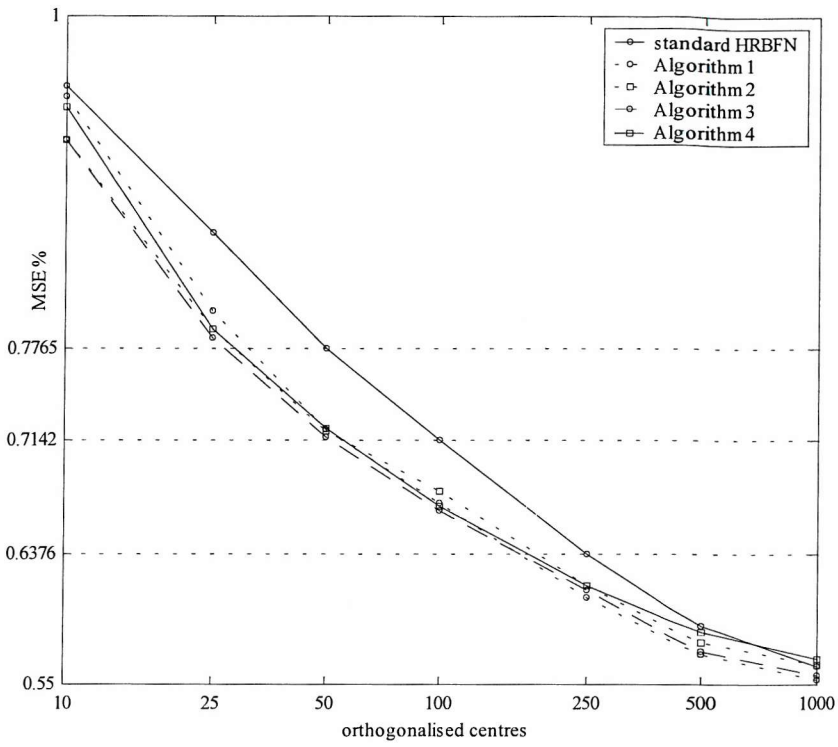

Figure 8. Comparison of MSE orthogonal sequences using the binary folding.

It is not easy to justify these observations rigorously, but we shall provide some conjectures to explain them. Firstly, there is the issue of the infinite support of the Gaussian kernels, and the resulting interaction between classes/clusters. As already stated, this mutual influence is changed as the input/centre space is folded. This might explain the small variations between different algorithms, since each technique moves the classes in a different way.

Secondly, many of the most significant centres may lie close to the set of fixed points. Clearly, these centres do not benefit the geometrical reduction given by folding. The study and identification of the fixed points is an interesting key to the optimisation of the algorithm. As we will see shortly, better results are obtained using a technique that in principle is equivalent to the clustering folding.

9.3.7. Moment-based classification of inputs

The previous two sections have developed an approach to extend the folding principle from the 2-dimensional case to an arbitrary number of dimensions. It has been shown

how the abstract properties of the folding principle can be applied to define a folding technique in a realistic case, the 8_1 sampling lattice.

In this section an alternative, more intuitive description of the folding principle is given, which will lead to a folding technique more general than cluster folding. Rather than focusing on the pixel-to-pixel relationships yielded by the flip operators, we will concentrate our attention on the change occurring in the luminance gradients when the input is flipped. Herein, this will be applied directly to the case of the 8_1 lattice.

Consider a generic input $\mathbf{x}$. The vertical flip $\mathbf{T}_V$ swaps the position of the pixels above the (horizontal) flip axis, specifically x_1, x_2, x_3 and x_4, with the position of the pixels below the flip axis, specifically x_5, x_6, x_7 and x_8. Consider a function that calculates a generic vertical mass gradient of $\mathbf{x}$, defined as:

$$\partial_V(\mathbf{x}) = (a_1 x_1 + a_2 x_2 + a_3 x_3 + a_4 x_4) - (a_5 x_5 + a_6 x_6 + a_7 x_7 + a_8 x_8) \tag{9.18}$$

with the constraint that all the weights a_i are positive. If the weights associated with pixels at the same distance from the horizontal axis are all equal, equation (9.18) represents the vertical moment of mass (around the horizontal axis), expressed as:

$$m_V(\mathbf{x}) = \mathbf{m}_V^T \mathbf{x} =$$

$$[m_{1V} \;\; m_{3V} \;\; m_{2V} \;\; m_{3V} \;\; -m_{3V} \;\; -m_{2V} \;\; -m_{3V} \;\; -m_{1V}]\mathbf{x} = \tag{9.19}$$

$$m_{1V}(x_1 - x_8) + m_{2V}(x_3 - x_6) + m_{3V}(x_2 + x_4 - x_5 - x_7)$$

The moment of mass can be considered as a measure of the average luminosity, and (9.19) is constructed as the weighted difference in luminosity above and below the horizontal axis, weighted according to the distance from the axis.

It is straightforward to see that $m_V(\mathbf{x}) = -m_V(\mathbf{T}_V \mathbf{x})$. Hence the vertical moment of mass changes sign according to the sign of the variant part of $\mathbf{x}$. It is easy to

demonstrate that the null space of $m_V(\mathbf{x})$, defined as $\mathbf{x} \in R^8 : m_V(\mathbf{x}) = 0$, includes the set $F_{\mathbf{T}_V}$ of the fixed points of $\mathbf{T}_V$. To show this, recall that $F_{\mathbf{T}_V}$ is defined as the set of inputs $\mathbf{x}$ such that $\mathbf{T}_V\ \mathbf{x} = \mathbf{x}$. For the 8_1 lattice, this implies that

$$\mathbf{x} \in F_{\mathbf{T}_V} \Rightarrow x_1 = x_8,\ x_3 = x_6, x_2 = x_5,\ x_4 = x_7 \tag{9.20}$$

Substituting (9.20) in (9.19), shows that $m_V(\mathbf{x}) = 0$. Note that (9.20) is more general than (9.19), since the condition $a_2 = a_5 = a_4 = a_7$ is not necessary.

One can equally define an horizontal moment of mass, relative to the horizontal flip operator $\mathbf{T}_H$ and the vertical axis of symmetry. However, the expression is simpler, since the pixels on the vertical axis do not appear:

$$m_H(\mathbf{x}) = \mathbf{m}_H^T\ \mathbf{x} = [0\ \ m_H\ \ 0\ \ m_H\ \ -m_H\ \ 0\ \ -m_H\ \ 0]\,\mathbf{x} = \tag{9.22}$$

$$m_H(x_2 - x_4 + x_5 - x_7)$$

In this case $m_H(\mathbf{x}) = -m_H(\mathbf{T}_H\ \mathbf{x})$, and again the condition of equal weights on the off-axis pixels is not necessary for the null space of $m_H(\mathbf{x})$ to correspond to $F_{\mathbf{T}_H}$. This condition becomes necessary if it is required that $m_H(\mathbf{x}) = m_H(\mathbf{T}_V\ \mathbf{x})$ and $m_V(\mathbf{x}) = m_V(\mathbf{T}_H\ \mathbf{x})$ in order to make the horizontal luminosity gradient independent of $\mathbf{T}_V$ and vice-versa.

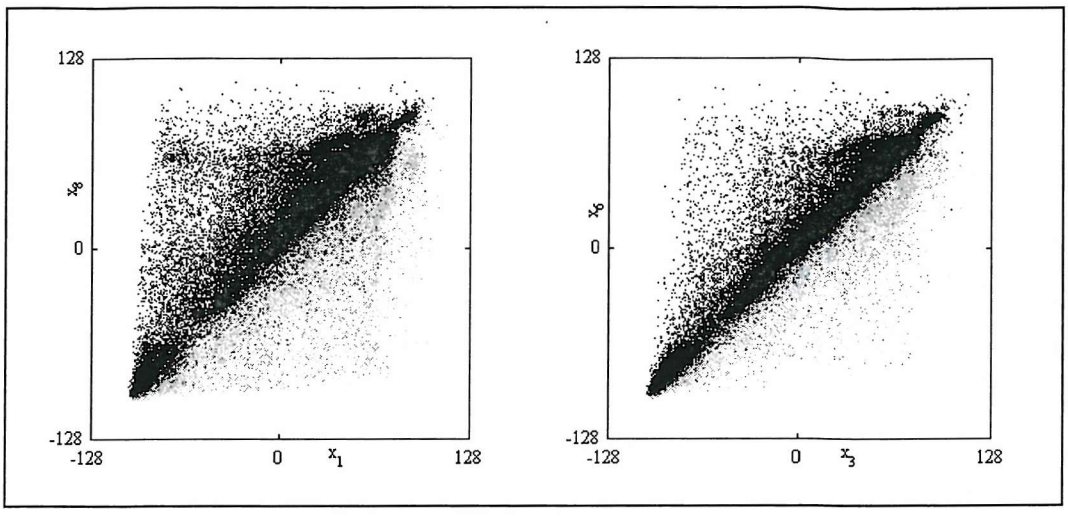

Figure 9. Projected planes: moment folded inputs (black) , original inputs (grey).

Figure 9 shows the distribution of inputs before and after moment folding in two of the projected planes. It is evident that there is no sharp cut in $x_1 - x_8$ as seen in cluster folding. This is because the cuts happen in planes orthogonal to the two moment vectors $\mathbf{m}_H$ and $\mathbf{m}_V$, not parallel to the planes shown in figure 9.

9.3.8. Results for moment folding RBFN

One is free to choose the three moment weights, but there are several alternatives that suggest themselves. Figure 10 shows the orthogonalisation plot of the moment-reduced network for 3 different choices: all weights equal to 1, $m_{2V} = 2$ and $m_{1V} = m_{3V} = 1$, $m_{1V} = 1$ and $m_{2V} = m_{3V} = 2$.

These results imply that moment folding generally outperforms binary cluster folding. Specifically, the plot clearly shows that the window of reducible centres is enlarged to $20 \sim 200$. Different trends may appear as the initial choice of centres is changed, and we can not definitely conclude that moment folding has a better performance than cluster folding. However in the region M>25 the two methods are similar and neither achieves the performance of a Volterra series (MSE=0.6956%) with M<50.

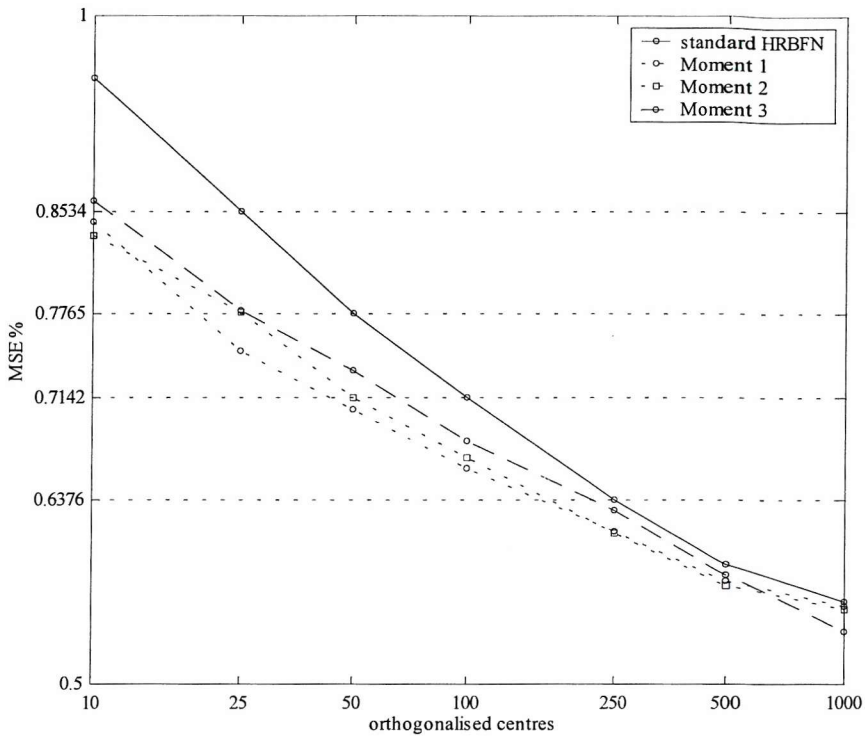

Figure 10. Comparison of MSE orthogonal sequences using the moment folding scheme.

9.3.9. Non-linear optimisation and folding

This section considers the application of the Nelder-Mead optimisation scheme to enhance the performance of the folded networks. The network is reduced to 10, 16, 20 and 25 orthogonal folded centres. The cluster folding algorithm based on the sign of $x_1 - x_8$ is applied. The 4 networks are successively optimised using the Nelder-Mead algorithm. The results are depicted in figure 11, where the MSE is plotted as the number of iterations of the NM algorithm is increased. As in standard RBFN, some form of non-linear optimisation improves the interpolator performance. However the curves tend to flatten as the number of iterations increases. Note that in this case the RBFN is able to achieve the same MSE as the Volterra model using only 16 centres. This is well within the range suggested in appendix B. As in chapter 6 the optimisation has driven some of the centres outside the boundaries of the input set (figure 12). This phenomenon again re-emphasises one limitation of the random initial selection of centres from the input space.

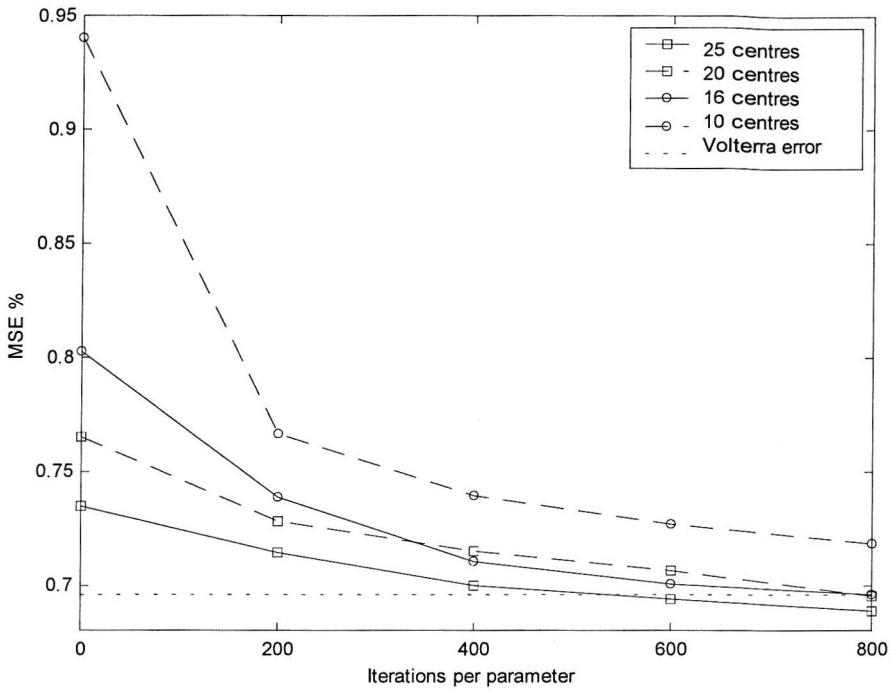

Figure 11. MSE sequences for 25, 20, 16 and 10 centres folded networks optimised using the Nelder-Mead optimisation algorithm.

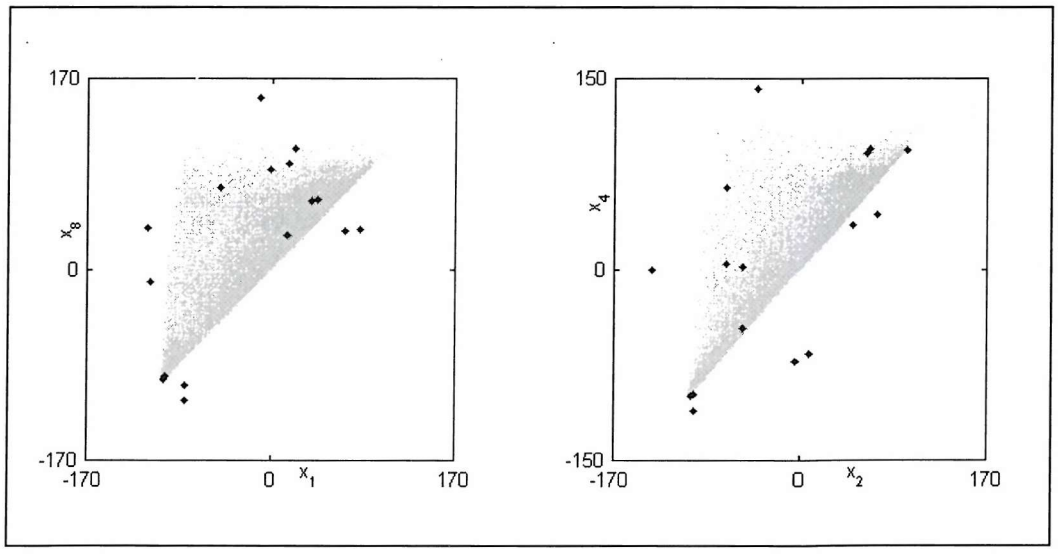

Figure 12. projected spaces of Nelder-Mead optimised symmetric networks (16 centres).

9.4. Conclusions

Chapter 8 showed that the symmetry properties of the linear interpolating function can be used to provide significant reductions in the computational cost without loss of performance. The built-in symmetry of the sampling lattice, and the conjectured rotational invariance of the local topology of the interpolation map, ensures that the optimal linear weight vector is orthogonal to a subspace of the input space. Hence the linear interpolating function can be re-arranged such that the number of multiplication units is reduced for a broad range of sampling lattices.

This chapter showed how these principles can be extended to exploit similar properties in RBFN. It has been shown that while the information contained in the orthogonal space is not relevant to the linear problem, RBFN use a significant portion of it to yield a better interpolation. However, the conditions of symmetry are derived from a set of general properties of image interpolation, and do not depend on the particular model. It is reasonable to expect that some form of symmetric reduction is possible for non-linear systems, and specifically for RBFN.

In the linear case the conditions of symmetry modify the geometrical properties of the $R^D \rightarrow R$ mapping. RBFN have a strong geometrical structure, and the set of centres can be regarded as a prototype of the distribution of the input set. It is reasonable to expect that symmetry modifies the properties of the $R^D \rightarrow R^M \rightarrow R$ RBFN mapping.

Symmetry was used to relate an input $\mathbf{x}$ with its flipped versions, since they are assumed to generate the same output. In the symmetric RBFN (section 9.2), each centre is used to generate a set of flipped centres. Consequently, the intermediate space can be divided into symmetric subspaces that can be reduced using linear techniques. The input distributions are generally symmetric, or quasi-symmetric, hence the symmetric RBFN is a valid model at least when the number of centres is high. Training is critical when the number of centres is small, since the condition of local symmetry may affect the performance of orthogonalisation techniques.

Experiments show that the proposed model is reasonably valid, but training should be studied more completely.

The main computational load in the RBFN is proportional to the number of centres M. A cost-saving strategy must aim principally at reducing the network branches, i.e. to reduce the number of centres. In this case it is not possible to derive simple analytical results, since the reduction of centres affects the non-linear part of the RBFN model.

The folding principle derives from an observation of the RBFN model in 2 dimensions, and considering simplified non-linear kernels with small support in R^2. An interesting consequence is that the input space is separable into two sub-spaces, which yield the same output dynamics. These sub-spaces have mirror symmetry along the bisector. The application of the flip operator to one of these sub-spaces makes it fold onto the other sub-space. Hence the area of the input space can be reduced by half without significantly affecting the dynamics. This reduction can be obtained either by discarding one of the two subspaces, or alternatively by folding the input space across the symmetry line. RBFN are generally based on infinite support kernels, so that some consequences are introduced. Basically, the set of centres cannot be folded without experiencing some loss in performance. This problem is discussed in section 9.3.1, where qualitative considerations suggest some mechanisms that generate these performance degradations. In sections 9.3.2 and 9.3.3 the folding principle is extended into spaces with an arbitrary number of dimensions. Two folding strategies are produced: the binary class folding method and the moment-based folding method.

The work concentrated on the practical case of a HGRBFN with lattice 8_1. The results show a significant reduction of the number of centres required to achieve a specified MSE. However, the simplistic expectation of a reduction by 4 (a factor 2 for each flip operator) is not matched by the results. It is necessary to improve the model in order to account for, and compensate for, any loss given by the missing cross-influence of the clusters, and the influence of centres near the locus of fixed points should be accounted for.

A simple application of Nelder-Mead optimisation shows again the limitations of the choice of centres from the input set. The optimisation algorithm is encouraged to recover the essential information missed in folding. Folding reduction, followed by Nelder-Mead optimisation, matches the performance of the 3rd order Volterra series in a complexity range where predictions suggest that folded RBFN is comparable to the Volterra series, in terms of computational load.

10. GENERALISATION OF SYMMETRIC ARCHITECTURES

10.1. Introduction

This chapter presents the results of studies into the generalisation ability of symmetric architectures, specifically the symmetric linear filter and the folded HGRBFN, both using the 8_1 sampling lattice. The investigation will make use of the same frame set and techniques used in section 7.

10.2. Linear results

Two aspects regarding the generality of the symmetric linear filter are considered. The first regards the ability of the filter to interpolate a rotated/flipped version of the training set. The second regards the cross-result over the complete frame set compared to the best result achievable on each frame.

In chapter 8 it was already pointed out that, at least for a flip around the vertical axis, corresponding to the flip operator $\mathbf{T}_H$, the flipped image can be considered as a valid image. On the other hand a horizontally rotated image ($\mathbf{T}_V$) more often looks unnatural, yet it can be conjectured that a realistic system should yield a uniform result on any possible flip or rotation of the training set.

Table 1 shows the results of applying non-symmetric linear filters to the flipped versions of their training sets, compared to the corresponding symmetric result (that, by construction, is identical over the 3 flip operators). The results are normalised with respect to the non-symmetric self-result (i.e. the result using the non-symmetric filter on the non-flipped frame). One can see that the symmetric filter produces a slightly worse result on the non-flipped training set than that achieved using the non-symmetric filter. This is indicated by the normalised result being greater than unity. This result is not a surprise, since the non-symmetric filter is not constrained in its interpolation by the conditions of symmetry. However, the symmetric filter performs better on any flipped version of the training set, and its error is below the average

result of the non-symmetric filter. It is therefore reasonable to conclude that the symmetric filter increases the generalisation ability of the filter.

		input frame rotation (non-symmetric filter)				
		H	V	HV	average	Symmetric filter
training frame	1	1.2669	1.2724	1.0066	1.1365	1.0652
	2	1.0342	1.0248	0.9904	1.0123	1.0095
	3	1.0089	1.0119	1.0029	1.0059	1.0022
	4	1.0582	1.0522	0.9937	1.0260	1.0144
	5	1.0020	1.0066	1.0046	1.0033	1.0011
	6	1.0121	1.0146	1.0083	1.0087	1.0031

Table 1. Error of the non-symmetric filters on flipped inputs, normalised with respect to the error on the non-flipped input. Error of the symmetric linear filters.

		Input frame					
		1	2	3	4	5	6
training frame	1	1.0000	1.0021	1.0112	1.0415	1.0028	1.0065
	2	1.0009	1.0000	1.0137	1.0308	1.0064	1.0022
	3	1.0066	1.0090	1.0000	1.0620	1.0029	1.0185
	4	1.0606	1.0565	1.1785	1.0000	1.1260	1.0350
	5	1.0025	1.0077	1.0063	1.0618	1.0000	1.0194
	6	1.0038	1.0026	1.0325	1.0204	1.0168	1.0000

Table 2.a. Symmetric linear filter cross-results, normalised with respect to symmetric self-results.

		Input frame					
		1	2	3	4	5	6
training frame	1	1.0652	0.9552	0.8096	0.9636	0.8672	0.8747
	2	1.0287	1.0095	0.9733	1.0149	0.9908	0.9829
	3	0.9869	0.9923	1.0022	0.9894	0.9961	1.0027
	4	1.0373	1.0108	0.9699	1.0144	0.9924	0.9794
	5	1.0051	1.0010	0.9968	1.0040	1.0011	0.9983
	6	0.9808	0.9875	1.0015	0.9838	0.9922	1.0031

Table 2.b. Symmetric/non-symmetric linear filters cross-result ratio.

Table 2.a shows the symmetric cross-results. One can see how the cross-results are always worse than the self-result. Table 2.b shows the ratio of the MSE obtained for symmetric and non-symmetric filters when trained on one frame and tested on a second. It can be seen that the symmetric filter generally offers slightly better performance when applied to frames it has not been trained on. In fact, one can see that the average deviation from the unity of the results in table 2.a is smaller than the

corresponding deviation in table 1.a in section 7.5.1, illustrating a more uniform behaviour.

From these tables, we can assess the effectiveness of the symmetric reduction as a generalisation technique. As well as a computational advantage, an improvement in generalisation is obtained. This improvement is due to a training procedure that forces the filter to produce uniform results over flipped versions of the training set. Moreover, this generalisation ability is built-in via a-priori conditions, which do not increase the computational load, unlike regularisation that typically requires the minimisation of extra terms. In fact, the computational load is reduced since there are fewer coefficients to calculate.

10.3. RBFN results

Figure 1 shows a comparison of the self-results for a folded and unfolded networks using 4 different folding strategies. All three results are normalised with respect to the non-symmetric linear result. Two binary folding techniques are used, the first using $sign(x_1 - x_8)$ as the decision threshold for the vertical flip, and the second using $sign(x_3 - x_6)$. Both techniques use $sign(x_2 - x_4)$ for the horizontal flip. Results for two moment folding techniques are also depicted, which use respectively $\mathbf{m}_V = [1\ 1\ 1\ 1\ -1\ -1\ -1\ -1]^T$ and $\mathbf{m}_V = [1\ 1\ 2\ 1\ -1\ -2\ -1\ -1]^T$ (closest neighbours moment, with x_3 and x_6 being emphasised) for the vertical flip. Both use the horizontal moment vector $\mathbf{m}_V = [0\ 1\ 0\ 1\ -1\ 0\ -1\ 0]^T$.

The plots clearly show results similar to those obtained in section 9. There is an approximate reduction by a factor of a half of the number of centres required to yield a given MSE, although the exact value varies from network to network (with an impressive reduction by 4 for network 5), and from strategy to strategy. One should remember that the training is initialised using a random selection strategy from an initially large number of centres (M=1000) and, as observed in chapter 6, the subsequent reduction may produce different results depending on this choice. However, it seems clear that the observed halving of centres can be considered as a general "rule of thumb".

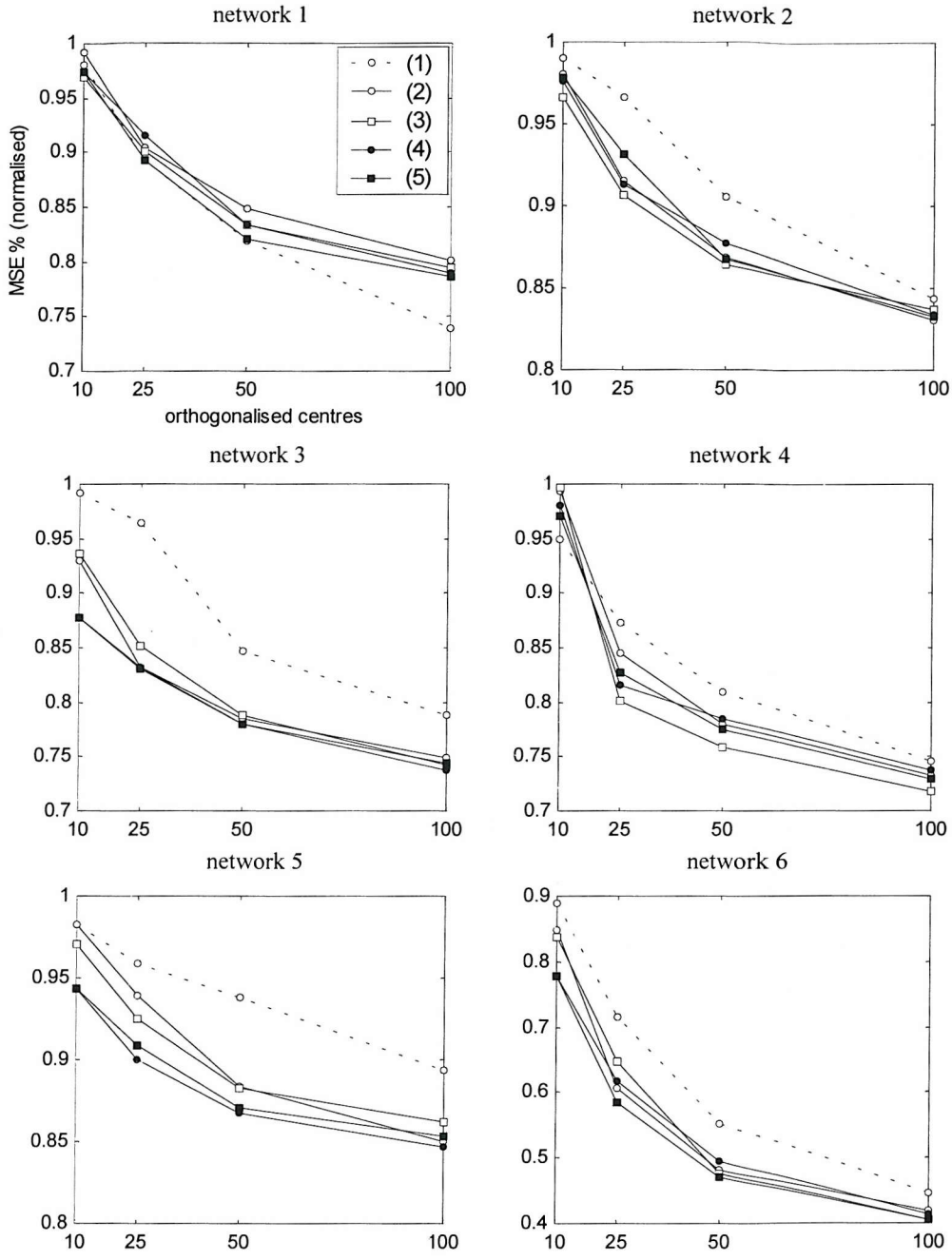

Figure 1. Folded self-results, compared to non-folded self-results. (1) non-folded OLS sequence. (2) binary folding using $sign(x_1 - x_8)$. (3) binary folding using $sign(x_3 - x_6)$. (4) moment folding, equal weights. (5) moment folding, closest neighbours. Results are normalised with respect to the linear non-symmetric self-result.

The behaviour of network 1 is rather peculiar. In this case, the folding techniques seem to degrade the performance, especially for M=100. This lack of performance may be the consequence of peculiar characteristics of the training set, rather than a general failure of the folding technique.

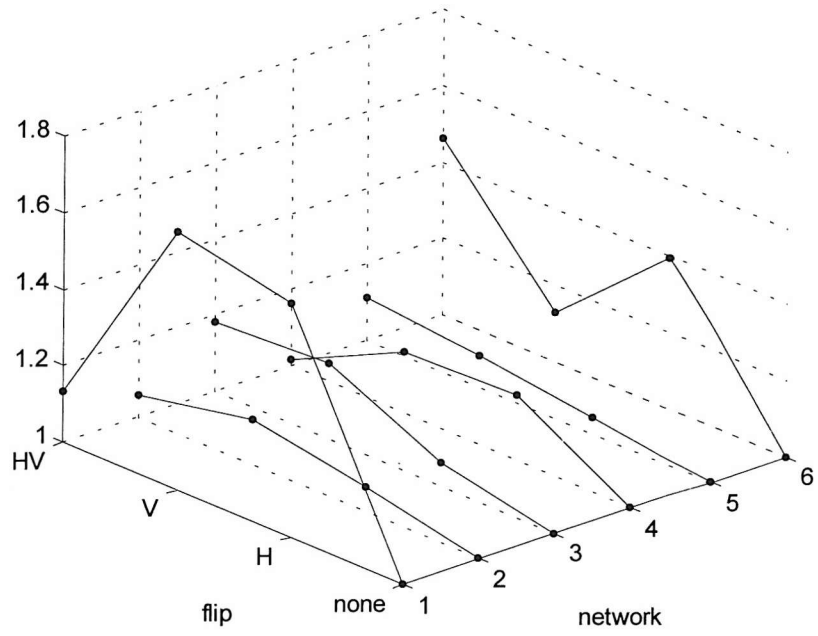

Figure 2.a. results over the flipped frames, unfolded networks 1 to 6, normalised with respect to the performance on the unfolded input.

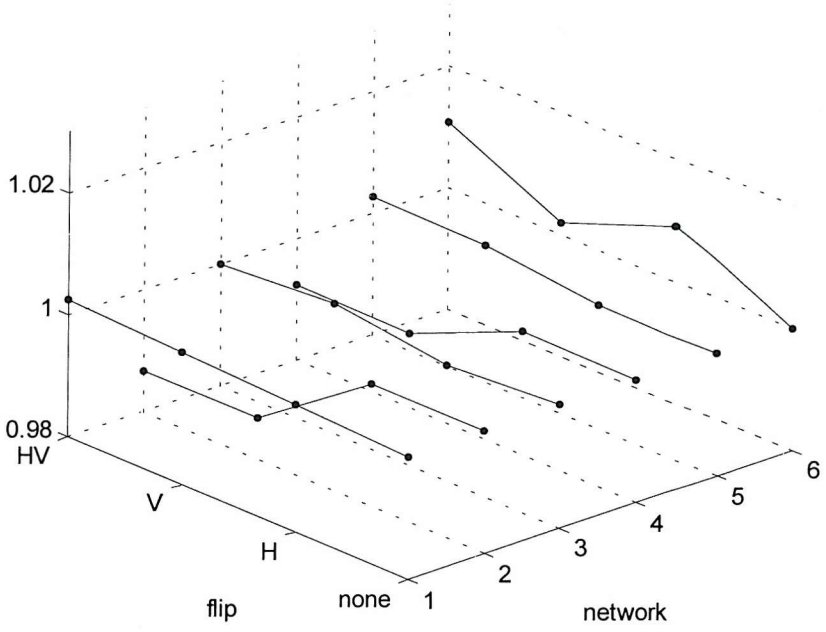

Figure 2.b. results over the flipped frames, equal weights moment folded networks 1 to 6, normalised with respect to the performance on the unfolded input.

Figure 2.a shows the MSE obtained when the training sets and their flipped versions are submitted as input to their corresponding non-symmetric (unfolded) network, using 100 OLS centres. The results are normalised with respect to the MSE obtained using the non-flipped set. One can see that network 1 exhibits a clear lack of generalisation, since the *H*-flip and *V*-flip versions of the training set produce significantly different results, whilst the *HV*-flip interpolation is more accurate. That might be explained by the presence of a dominant directional feature in the frame, invariant to the *HV*-flip. Note that similar behaviour is also shown by network 6. This might be explained by the presence of a directional feature invariant to the *H*-flip. Conversely, the corresponding equal weight moment folded network exhibits a quasi-uniform outcome over the 4 versions of the training set (Figure 2.b.). We will shortly see how the folded version of network 1 produces more general results than the unfolded one. One can see from figure 2.b. that the folded network performance is not absolutely constant over the 4 flipped versions of the input. In the linear case, the symmetric filter yields an exactly constant result by construction. In the RBFN case, the folding strategy does not yield a constant result due to the fact that the folded basis functions have large support. Therefore folding strategies do not produce exact symmetrisation.

10.3.1. RBFN cross-results

This section discusses the cross-results over the 6 frames, using 100 centres and 50 centres folded networks. Figures 3.a and 3.b show the plots for 50 centres folded network 1 to 6 compared to the 100 centre unfolded networks, with MSE normalised with respect to the unfolded self-result. In figures 3.c. and 3.d. equivalent results using 100 centres folded networks are plotted. Note that the unfolded result is highlighted by the dotted line for clarity. One can see how the results vary from network to network, and how the different folding strategies act in different ways. Some general conclusions can be drawn from these plots. It is clear that, with the proper choice of network and folding technique, it is possible to improve the generalisation ability of a network. At the same time one can produce a similar error on the self-result with half the number of centres. In particular it appears that the closest-neighbour moment folding produces, on average, best performance both in terms of MMSE and generalisation.

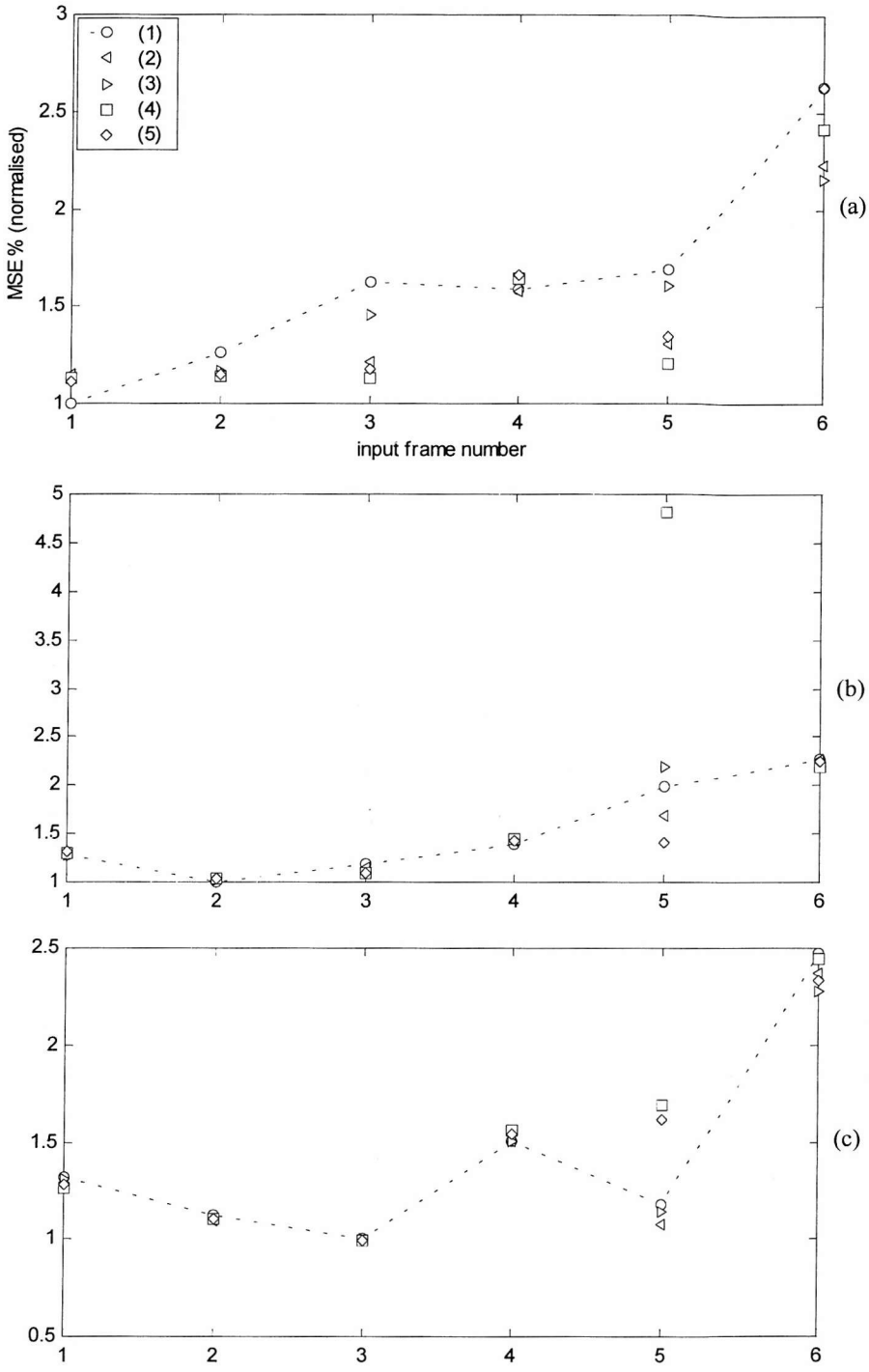

Figure 3.a. 50 centres folded cross-results, compared to 100 centres unfolded cross-results, normalised with respect to 100 centres unfolded self-results. (1) unfolded result (2) binary folding on $sign(x_1 - x_8)$ (3) binary folding on $sign(x_3 - x_6)$ (4) equal weights moment folding (5) closest neighbour moment folding. (a) network 1 (b) network 2 (c) network 3.

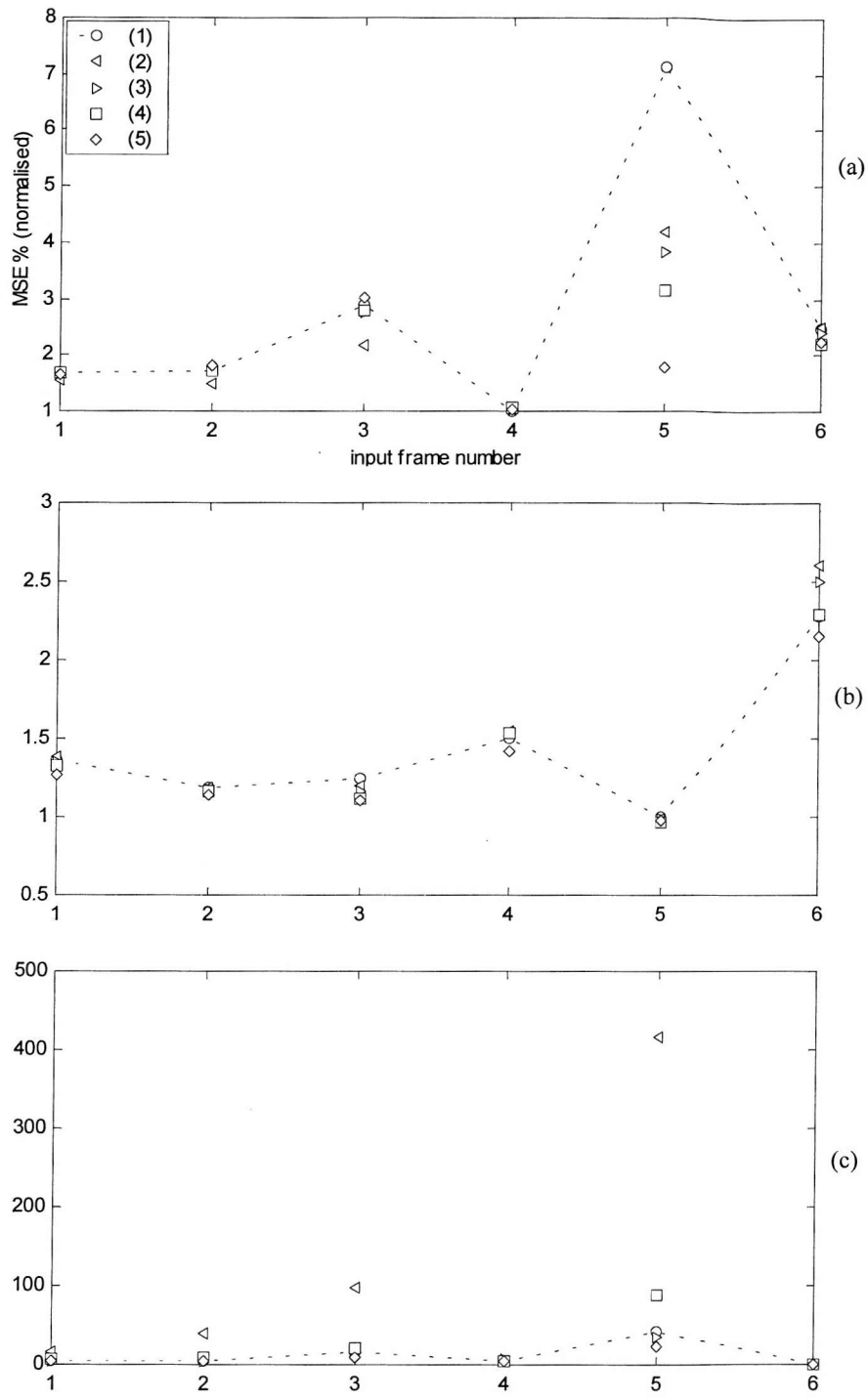

Figure 3.b. 50 centres folded cross-results, compared to 100 centres unfolded cross-results, normalised with respect to 100 centres unfolded self-results. (1) unfolded result (2) binary folding on $sign(x_1 - x_8)$ (3) binary folding on $sign(x_3 - x_6)$ (4) equal weights moment folding (5) closest neighbour moment folding. (a) network 4 (b) network 5 (c) network 6.

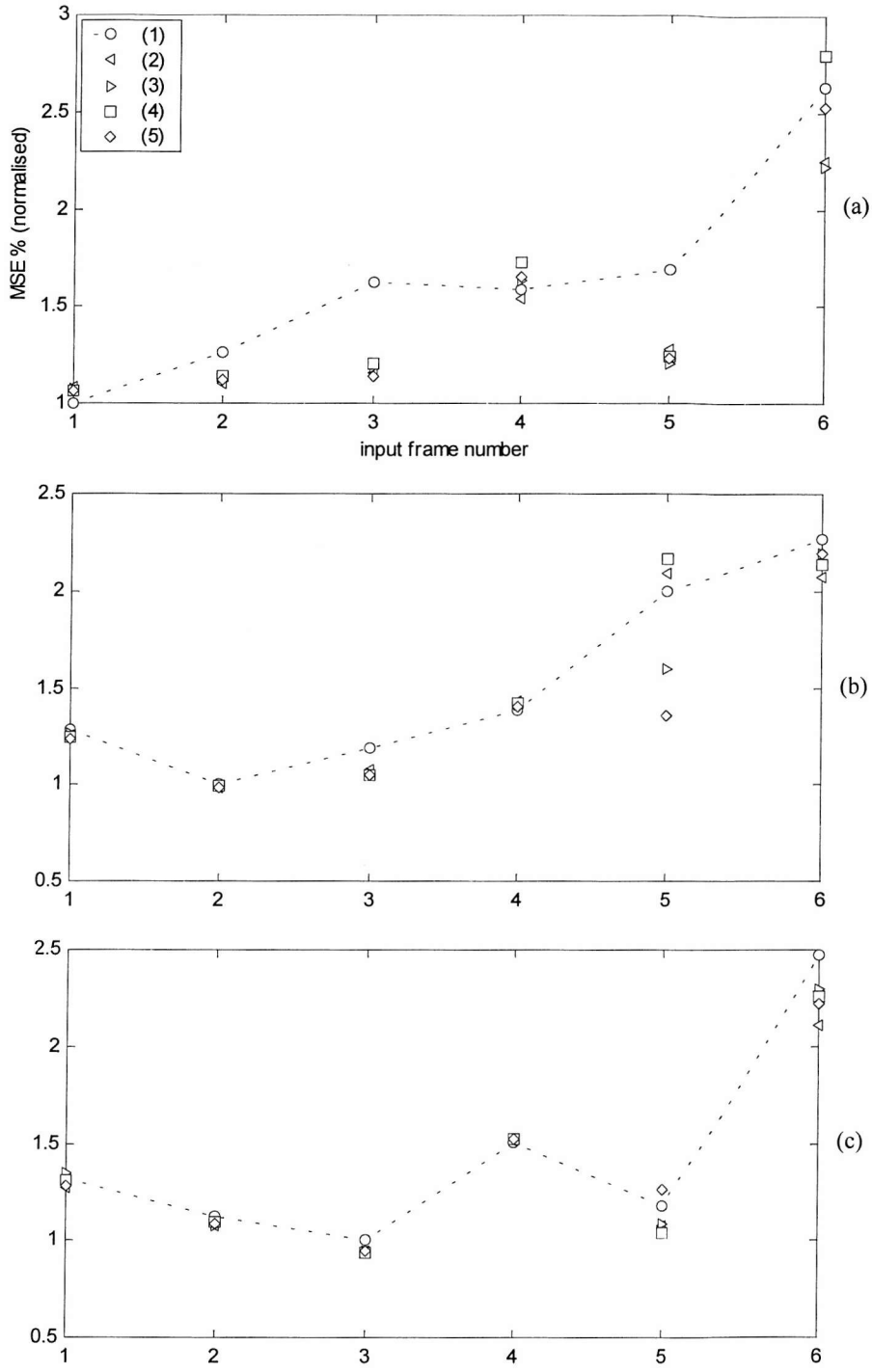

Figure 3.c. 100 centres folded cross-results, compared to 100 centres unfolded cross-results, normalised with respect to 100 centres unfolded self-results. (1) unfolded result (2) binary folding on $sign(x_1 - x_8)$ (3) binary folding on $sign(x_3 - x_6)$ (4) equal weights moment folding (5) closest neighbour moment folding. (a) network 1 (b) network 2 (c) network 3.

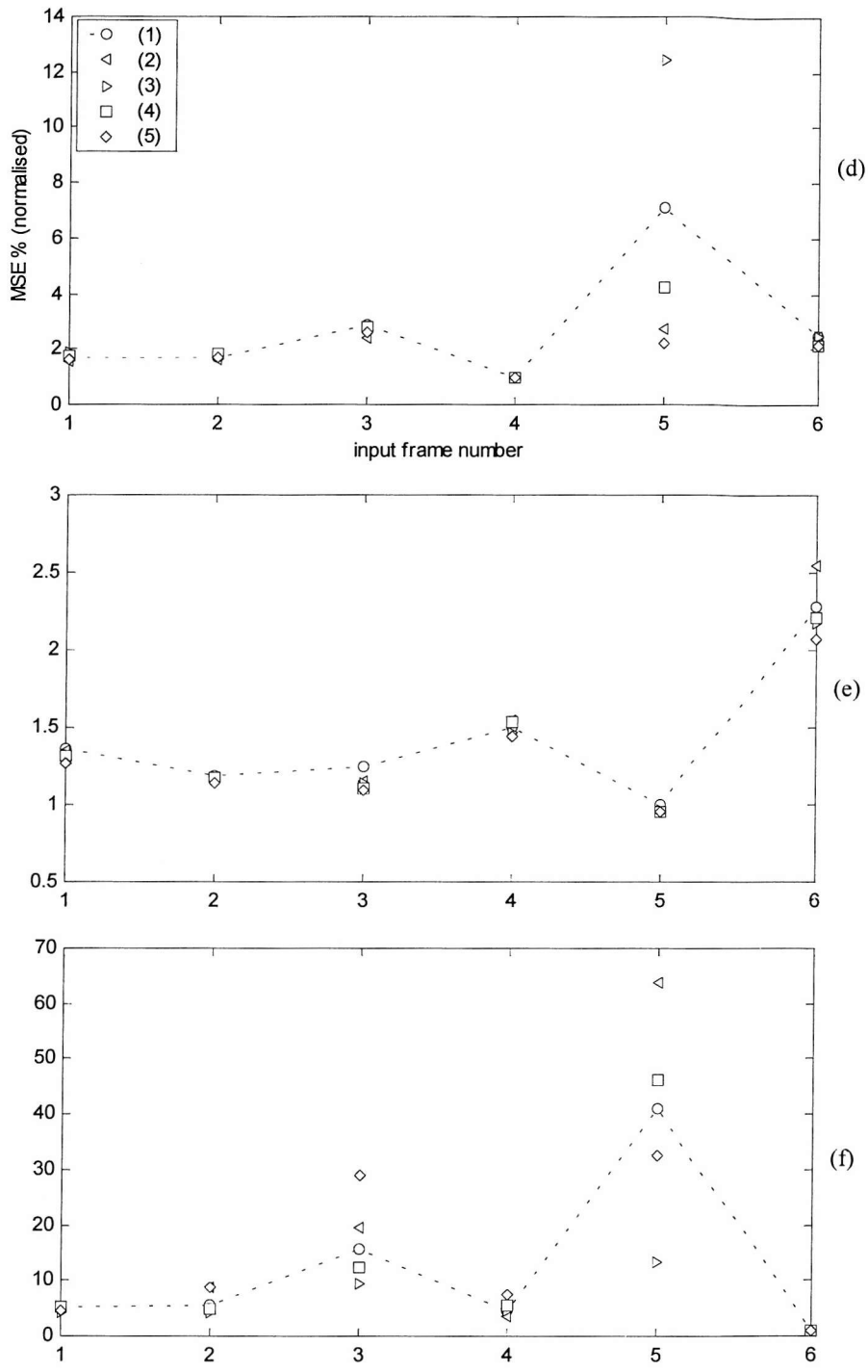

Figure 3.d. 100 centres folded cross-results, compared to 100 centres unfolded cross-results, normalised with respect to 100 centres unfolded self-results. (1) unfolded result (2) binary folding on $sign(x_1 - x_8)$ (3) binary folding on $sign(x_3 - x_6)$ (4) equal weights moment folding (5) closest neighbour moment folding. (d) network 4 (e) network 5 (f) network 6.

10.3.2. Bias and Variance

The top and middle plots in figure 4 illustrate the squared bias and variance of the closest-neighbour folded network trained on frame 2, plotted along with the results for the corresponding unfolded network. Despite the simplistic technique used to calculate the bias and variance (section 7.5.5), one can gain insights into the way folding techniques works. Note how the variance is approximately halved using the folding technique. This explains the improved generalisation of the folded network. A reduced variance implies that the 6 folded mappings are close to each other.

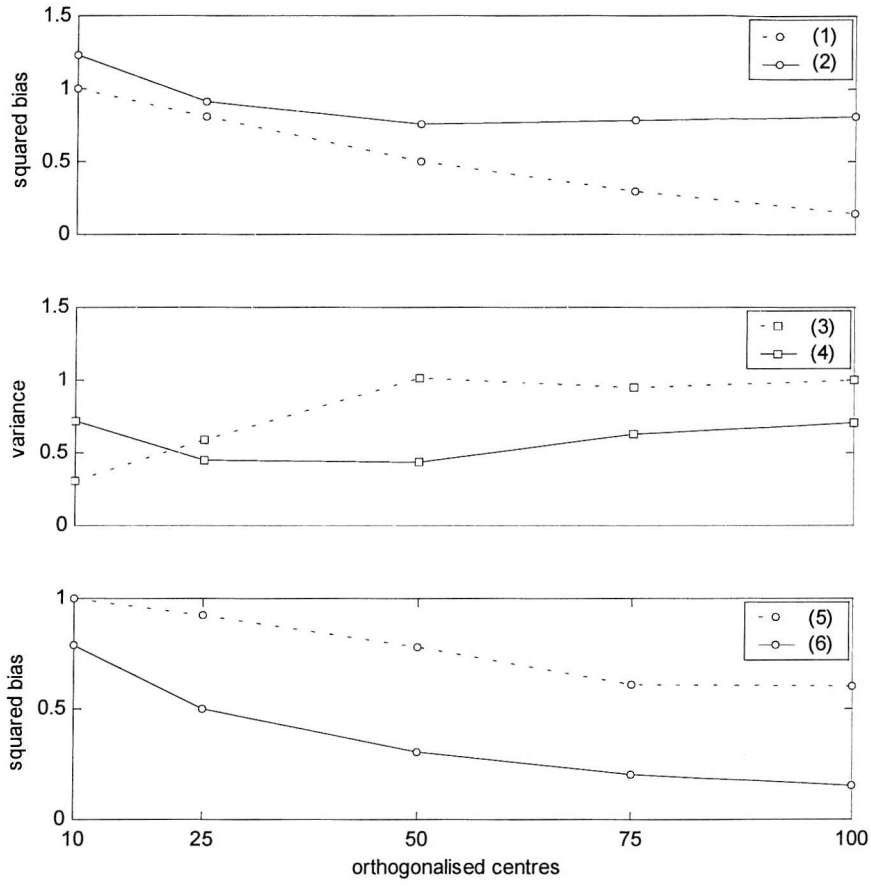

Figure 4. Squared bias and variance, unfolded and closest neighbours moment folded network 2. Unfolded (1) and folded (2) squared bias. Unfolded (3) and folded (4) variance. Unfolded (5) and folded (6) squared bias using a folded model to estimate $\langle t | \mathbf{x} \rangle$.

One can also see a corresponding degradation in the bias, which flattens for M>50. This increase in the bias is unexpected, since the MSE yielded by the folded networks is typically lower than for their unfolded counterparts. However, one should realise that $\langle t|\mathbf{x}\rangle$ is calculated using an unfolded network. Therefore, $\langle t|\mathbf{x}\rangle$ may account for asymmetries in the training set that are primarily cancelled by the folded techniques and, in fact, represent a nuisance factor since the mapping should abide by the conditions of symmetry. Therefore it is assumed that the lower bias of the unfolded training is mainly due to the over-fitting of the unfolded $\langle t|\mathbf{x}\rangle$ on asymmetric features.

The bottom graph in figure 4 shows how, using a folded model to estimate $\langle t|\mathbf{x}\rangle$, the folded network yields lower bias and variance than its unfolded counterpart. Figure 5 shows that $\langle t|\mathbf{x}\rangle$, calculated using the folded model, produces results comparable to the unfolded $\langle t|\mathbf{x}\rangle$, in terms of individual results and generalisation. Therefore, we can assume that it is the bottom graph in figure 4, rather than the top one, which truly represents bias.

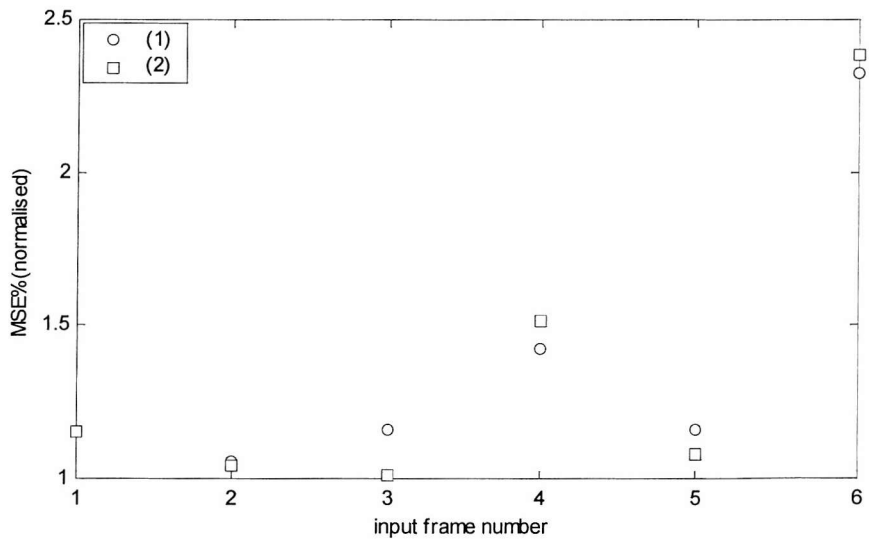

Figure 5. Estimate of $\langle t|\mathbf{x}\rangle$. (1) Evaluated with 100 centres, unfolded network. (2) Evaluated with 100 centres, folded network. Plots normalised with respect to self-results.

10.3.3. Weight-decay training

Figure 6 shows the results of weight-decay training (sections 7.4.1 and 7.6), applied to the closest-neighbour moment folded networks with 50 centres, compared to their unfolded version using 100 centres. The plots show the average, maximum and minimum value over the whole frame set. The results are normalised with respect to the non-regularised, unfolded self-result (using 100 centres).

One can see that weight-decay has an effect on folded networks similar to its effect on unfolded networks. In particular in the case of network 2, one can see that the initial maximum value for the folded network is higher, but regularisation rapidly reduces this value. Table 3 shows how the sum-of-squared-weights value changes with the network considered, and with the folding technique. In particular, one can see that the 50 centres, closest-neighbour folded network 2 has a greater sum-of-squared-weights value, compared to its 100 centres unfolded counterpart. This may explain the initially poorer results seen in figure 6.

It is interesting to discuss, albeit in a qualitative fashion, the relationship between symmetrisation and weight-decay in the linear case. Figure 7 illustrates three 2-dimensional examples. With two weights, w_1 and w_2, coming from a vertical symmetric sampling lattice, the symmetric solution fixes $w_1 = w_2$, so the solution is constrained to lie on the main diagonal of the weight space. On the other hand, a regularised solution is such that the sum of squared weights is minimised together with the sum-of-squared errors. The curves $w_1^2 + w_2^2 = const$ trace circles in the weight space centred in the origin. It is clear from this figure that the symmetric sum-of-squared-weights value will be greater or smaller than the non-symmetric solution depending upon the orientation of the quadratic error surface with respects to the main diagonal.

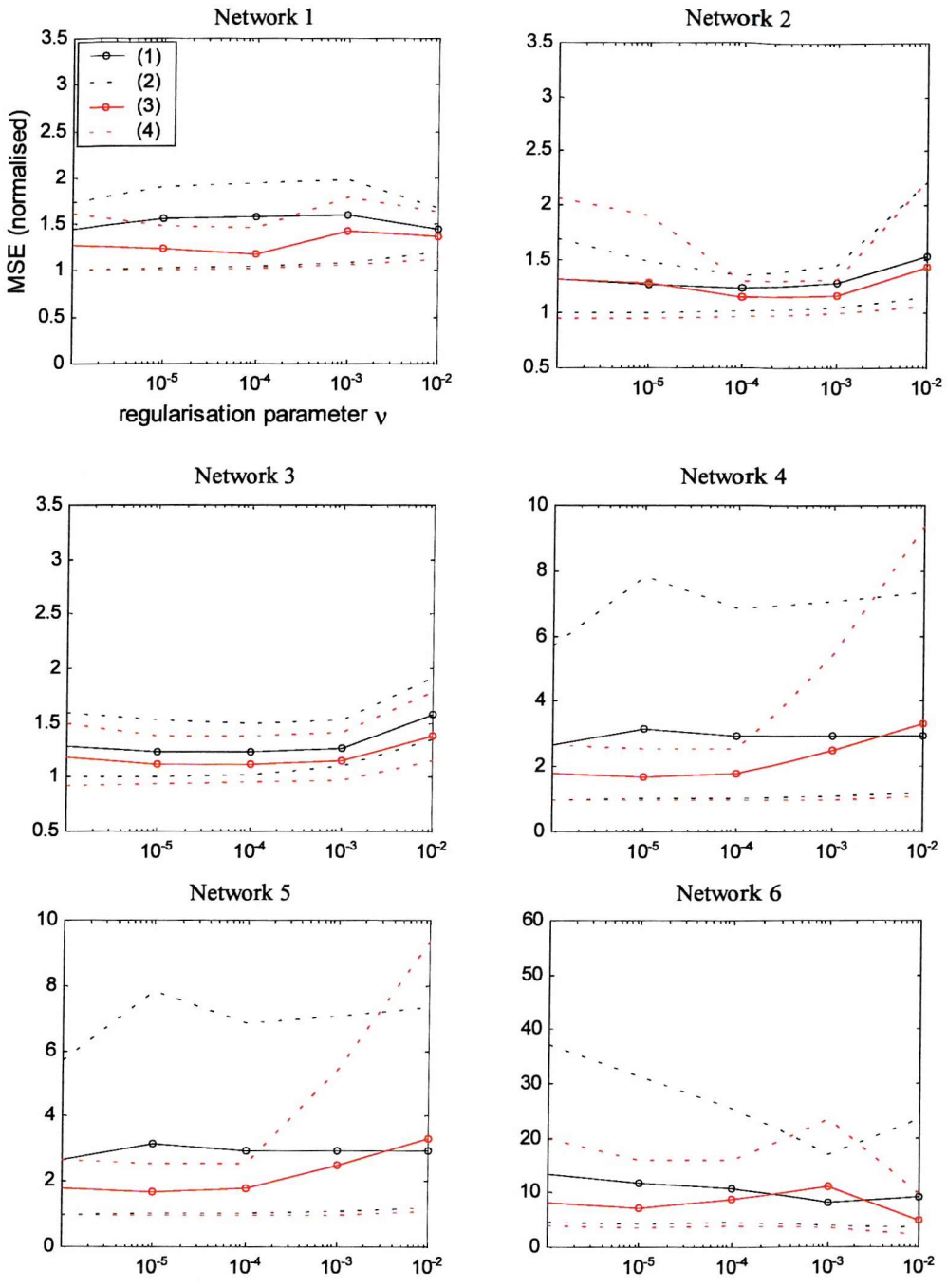

Figure 6. Weight decay training of folded networks (red) compared to unfolded networks (black). Results are normalised with respect to non-regularised unfolded results. (1) average over the frame set, standard network (2) minimum and maximum values over the frame set, standard network (3) average over the frame set, symmetric network (4) minimum and maximum values over the frame set, symmetric network.

		M	unfolded	equal weights	c. neighbours	x_1-x_8	x_3-x_6
Network	1	50	7.8504	2.4802	2.2976	4.3134	7.0287
		100	4.2493	5.6976	15.6113	13.2129	29.0995
	2	50	2.7868	1.7814	2.5185	1.8030	0.5094
		100	1.5085	2.9628	3.2271	3.5246	5.1097
	3	50	5.4700	6.3237	4.0356	1.5886	2.4275
		100	2.9608	3.9306	3.1315	3.0229	3.1065
	4	50	5.3989	3.3101	0.8241	2.6180	3.7219
		100	2.9223	4.0390	4.1367	5.6610	6.5139
	5	50	6.1225	2.7696	3.7347	5.3452	2.1168
		100	3.3140	1.7185	3.5280	2.9461	4.0320
	6	50	8.9955	2.4041	5.4389	6.7705	3.1059
		100	4.8691	7.8743	16.6147	11.4006	7.1085

Table 3. Sum-of-squared weights for unfolded and folded networks, normalised ($10^{-4} \cdot \mathbf{\Phi}^T \mathbf{\Phi}/M$).

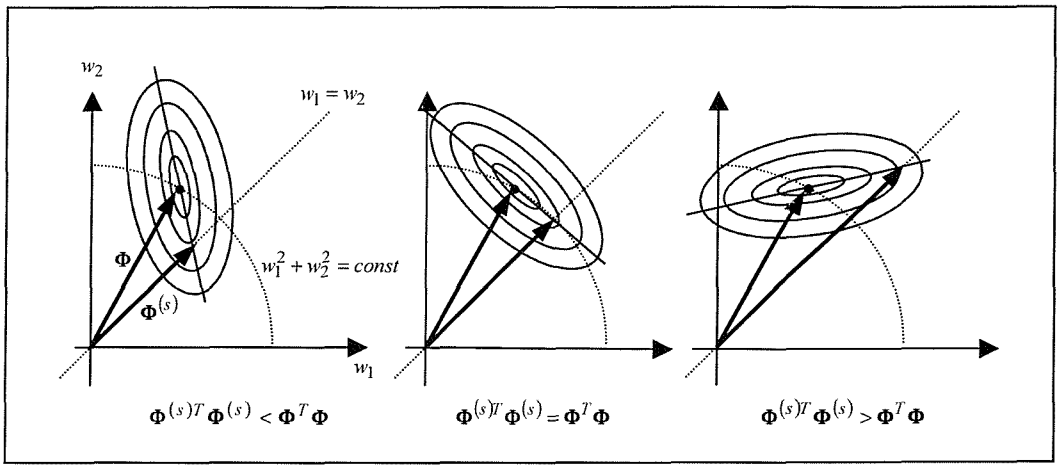

Figure 7. Symmetric solution and weight decay. The symmetric MMSE solution $\mathbf{\Phi}^{(s)}$ is found on the intersection of the quadratic error function with the bisector $w_1 = w_2$. The non-symmetric solution $\mathbf{\Phi}$ and $\mathbf{\Phi}^{(s)}$ lay on the direction identified by the smallest eigenvalue of the Hessian matrix (minimum slope). The squared sum of weights is increased or decreased by symmetrisation according to the orientation of the quadratic function.

10.3.4. Extended training on frame 6

Another point in common between the generalisation behaviour of unfolded and unfolded techniques is the poor performance on frame 6. In this section the extended training in section 7.7 is developed for this case. The initial set of centres is drawn from frame 2, folded using the closest neighbour moment technique and orthogonalised on frame 2, and the weights are recalculated using a linear combination of the cross-correlation matrices (equation 7.11). The results, compared to a 100 centres unfolded extended training, are shown in figures 8.a and 8.b, for 50 and 100 folded centres, respectively.

Using 50 folded centres, the results are practically indistinguishable from those obtained with the 100 centres unfolded network. Not surprisingly, the 100 centres folded network in fig. 8.b produces superior results.

From figure 9 one can see the generalisation performance of the extended training ($\nu = 0.15$) of the 50 centres folded network 2. Constraints were also added on the main diagonal of the input space, in order to reduce the degradation of the performance on frame 5 (see section 7.9). The result is compared to a 100 centres unfolded network, trained on frame 2 only, and to a 100 centres unfolded network trained using the constrained, extended training. It is evident that the folding technique offers comparable performance in terms of MSE and generalisation using half the number of centres and proves to be an effective and convenient reduction technique.

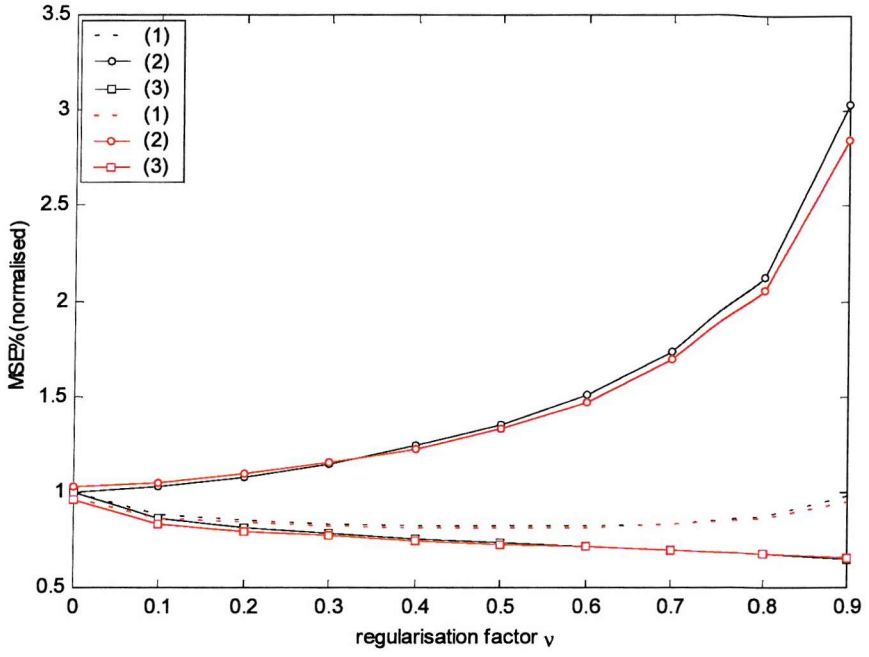

Figure 8.a. Extended training of network 2, 100 centres unfolded (black) and 50 centres folded (red) networks. (1) combined performance on training sets 2 and 6. (2) performance on training set 2 (3) performance on training set 6. Plots are normalised with respect to the unfolded initial values ($\nu = 0$).

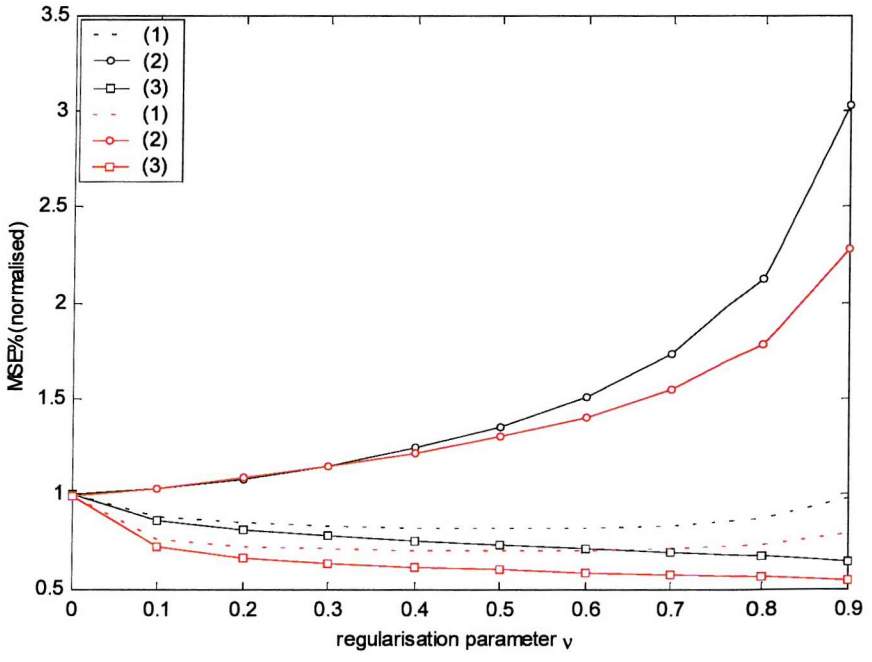

Figure 8.b. Extended training of network 2, 100 centres unfolded (black) and 100 centres folded (red) networks. (1) combined performance on training sets 2 and 6. (2) performance on training set 2 (3) performance on training set 6. Plots are normalised with respect to the unfolded initial values ($\nu = 0$).

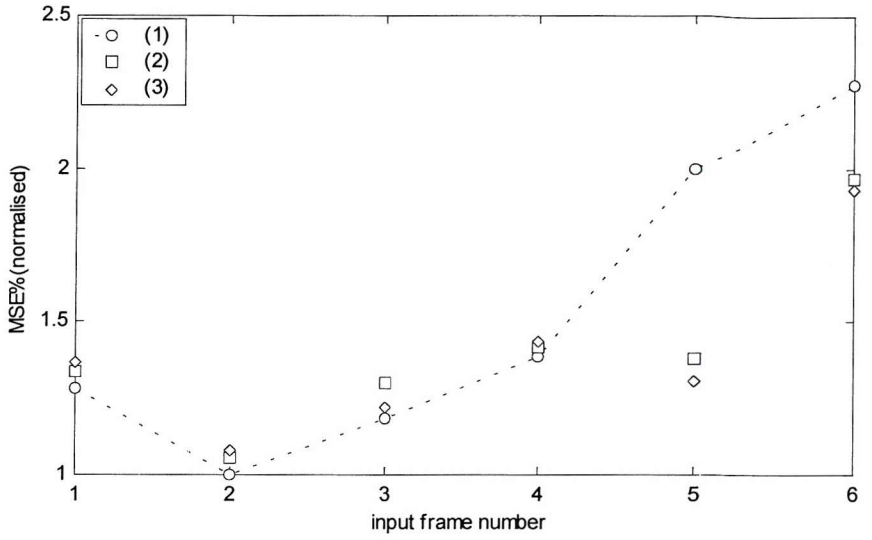

Figure 9. Extended training ($v = 0.15$) and constrained optimisation of network 2, over the frame set. (1) 100 centres unfolded network, unconstrained, non-extended. (2) 100 centres unfolded network, constrained, extended. (3) 50 centres folded network, constrained, extended.

10.4. Conclusions

The preceding two sections have developed a novel approach to the problem of reducing the complexity of an interpolation scheme based on RBFN. This approach addresses some rotational invariant features that standard RBFN training does not consider. By constructing systems that account for these features, these redundancies can be eliminated. In the case of a linear interpolator, such a problem has an exact and easily tractable solution. In RBFN case, a mixture of heuristic considerations regarding the geometry of the problem is used to obtain a convenient reduction in the number of centres necessary to yield a given MSE.

In this section, it has been shown that symmetrisation techniques also exhibit superior generalisation properties. Forcing the network to produce similar results on rotated versions of the same training set help to avoid over-fitting of directional features that cannot be considered as general properties of images. Moreover by using half the number of centres one can obtain the same results on a set of test frames. Another quantitative proof of this amelioration is given by the simultaneous reduction of bias and variance.

Clearly, there are some points that ought to be further investigated in the application of folding techniques. The most interesting of all is the relation between the reduction achieved and the topology of the input space. It has already been seen that an input space whose inputs are concentrated on the main diagonal (broadly corresponding to an image with few high-frequency components) might not benefit from the application of the folding techniques. On the other hand, there is little to be gained by using a non-linear technique for such an input space. A more accurate analysis would involve the study of the spaces of fixed-points $F_{\mathbf{T}_V}$, $F_{\mathbf{T}_H}$ and $F_{\mathbf{T}_{HV}}$, and how inputs and centres relate to these spaces where the folding technique is less effective.

Finally, non-linear minimisation techniques, which have not been investigated in the context of generalisation, should also be applied to obtain further reductions of the error. Such techniques should be used taking appropriate measures to avoid an excessive specialisation and consequently a decrease of generalisation.

11. CONCLUSIONS

The first part of this thesis (chapters 1 to 7) have shown how non-linear techniques produce significant improvements over linear filters when applied to solve the de-interlacing problem.

We have seen how de-interlacing can be considered a non-linear problem, mainly because in the process of interlacing there is a significant amount of information that is not registered, and cannot be recovered using linear methods since it constitutes a form of aliasing. The consequence is that linear filters typical produce a blurring artefact, known as "jagging", that is particularly noticeable on edges and detailed areas of an image. Another problem related to the application of linear filters is that they have insufficient complexity to deal with different situations. It has been shown how linear systems, trained on "natural" images, basically compute the average of the neighbouring pixels in order to estimate the unknown target pixel. This result explains the "jagging" artefact, since although an average filter produces good interpolation of uniform and smooth areas, unsatisfactory results are produced on edges and detailed areas.

The application of non-linear techniques mitigates this problem by inferring more complex relationships between neighbouring pixels, so that non-linear filters can deal with more complex situations. The performance on edges and detailed areas is better, and accordingly so is the perceptual quality of the interpolation. In this work we have investigated the application of two non-linear techniques, the Volterra series and the Radial Basis Function Networks, specifically focusing on the latter.

Both techniques deliver effective results, but this comes at the cost of increased computational complexity. This is a critical point in de-interlacing, since most de-interlacing systems are implemented in hardware and usually operate under real-time conditions. Therefore, a great effort must be placed in trying to reduce the computational cost. The comparison between Volterra series and RBFN has been conducted considering their relative computational cost, on the basis of some simple, yet realistic, considerations developed in appendix B. It has been shown that, with a

careful training, the two techniques can be considered equivalents in terms of results and costs.

Another problem that must be tackled when dealing with interpolation techniques is the ability to produce general systems, i.e. systems that are able to produce reasonable performance over a set of images much larger that the training set. This ability is called generalisation. Chapter 7 deals with this problem, and it is shown how an improper training leads to very specialised (i.e. non general) results. The problem is that the increased degrees of freedom given by non-linear techniques are spent to specialise the system on the particular training image. Therefore the training must be constrained somehow in order to avoid an excess of specialisation without affecting the error performance. A well known technique, weight-decay training, has been used to produce results that trade-off accuracy for generality. This has produced more general results, preserving the advantages of non-linear techniques. However, we believe that the problem cannot be completely solved using a single system, and suggested that the problem could be better tackled using a mixture of differently specialised systems.

The second part of this thesis (chapters 8 to 10) introduces a new RBFN training paradigm for image interpolation, the symmetric training. Image interpolation is typically performed using spatially symmetric sampling lattices, in which case, it is possible to devise equivalence relationships between vectors in the input space. In the linear case, the consequence is that the algebraic form of the filter can be simplified, and the computational load reduced.

Unfortunately, it is not possible to devise the same conclusions for non-linear techniques, since the algebra is non-linear. However, RBFN have a strong relationship to the topology of the input space, since the distribution of centres can be considered as a "copy" of the distribution of the input vectors. Therefore, a relationship between input vectors can be transferred to the centre vectors. We have described two techniques that are based on this principle. The symmetric RBFN successfully attempts to produce a result similar to the linear case, by inferring the same properties and reducing the number of linear weights in the RBFN. This does not produce large computational savings, as the greatest computational burden in the RBFN occurs the

non-linear layer. The folded RBFN directly tackles the non-linear problem by inferring symmetry constraints in the non-linear layer of the network. Albeit the analysis is more heuristic than in the symmetric RBFN case, nevertheless the results are satisfactory, and the computational load is reduced, on average, by half. Furthermore, the folded RBFN model includes the symmetric model.

Another advantage of exploiting symmetries in de-interlacing is that symmetrisation can be considered as a form of generalisation. By constraining the system to produce the same result on different areas of the input space, we obtain a more general result. The constraint of symmetry can be explained in this way: given an image, the system must produce equal (or similar) results on differently orientated versions of this image. This is equivalent to force the system to produce the same result on different images, and therefore the generality of the result is increased. Chapter 10 shows how symmetrisation is not only an effective reduction technique, but also a generalisation technique.

A. FOURIER SPECTRA OF THE VIDEO SIGNAL

A.1. Introduction

This appendix considers the spectrum of a video signal. The results obtained are applied in chapter 1 to draw the schematic plots of progressive and interlaced video, and to highlight their differences.

A.2. Continuous video signal and spectrum

A hypothetical continuous video signal can be expressed as a continuous sequence in time of continuous still images. Therefore, the signal is a function of two spatial co-ordinates x and y and a time co-ordinate t, $F_c(x,y,t)$, where conventionally x represents the horizontal direction, and y represents the vertical direction. Under the usual convergence conditions $F_c(x,y,t)$ can be Fourier-transformed into a frequency domain consisting of two spatial frequencies f_x, f_y and one temporal frequency f_t

$$\Phi_c(f_x,f_y,f_t)=\iiint F_c(x,y,t)exp\left[-2\pi i\left(f_x x+f_y y+f_t t\right)\right]dx\,dy\,dt \qquad (A.1)$$

A.3. Discrete video signal

Any real video sequence can be viewed as a sampled version of $F_c(x,y,t)$. The sampling spaces in each co-ordinate x, y and t are determined by a variety of circumstances. The consequence of this sampling is that the spectrum assumes a more complex form than that expressed by (A.1). From the sampling theorem, valid for any number of dimensions, the spectrum will be typically formed by an infinite series of replicas (aliases) of (A.1). The position of these aliases will be determined by the particular shape and structure of the sampling. For instance, if the sampling is uniform in space and time, the discrete signal can be expressed as:

$$F_d(n,m,k)=\sum_n\sum_m\sum_k F_c(x,y,t)\delta(x+n\cdot\Delta x)\delta(y+m\cdot\Delta y)\delta(t+k\cdot\Delta t) \qquad (A.2)$$

the spectrum of which is related to the continuous version by:

$$\Phi_d(f_x, f_y, f_t) = \sum_q \sum_r \sum_s \Phi_c\left(f_x - \frac{q}{\Delta x}, f_y - \frac{r}{\Delta y}, f_t - \frac{s}{\Delta t}\right) \qquad \text{(A.3)}$$

However, in many video applications the sampling lattice is more complicated than a simple, uniformly spaced one. Interlaced video represents one example, since the position of the lattice in the vertical direction is displaced by one row on adjacent time instants. To obtain an expression for the discrete spectrum it requires a more sophisticated analysis than the simple application of the sampling theorem, as in (A.3). The next sections will consider the Fourier spectrum of a continuous signal sampled on an arbitrary lattice.

A.4. Arbitrary lattice sampling

In order to produce the desired Fourier analysis, an arbitrary sampling of a continuous image is defined through a three dimensional relation between the continuous variables x, y, and t, and the corresponding discrete variables n, m, and k. These relations can be summarised as

$$\begin{aligned} x &= v_{11}\, n + v_{12}\, m + v_{13}\, k \\ y &= v_{21}\, n + v_{22}\, m + v_{23}\, k \\ t &= v_{31}\, n + v_{32}\, m + v_{33}\, k \end{aligned} \qquad \text{(A.4a)}$$

that can be described in a more compact form as:

$$\mathbf{x} = \mathbf{V}\,\mathbf{n} \qquad \text{(A.4b)}$$

where $\mathbf{x} = [x \;\; y \;\; t]^T$ and $\mathbf{n} = [n \; m \; k]^T$. It is useful (as will be clear shortly) to express the matrix $\mathbf{V}$ in terms of column vectors, $\mathbf{V} = [\mathbf{v}_1 \;\; \mathbf{v}_2 \;\; \mathbf{v}_3]$. It is therefore possible to express the sampling (A.2) in matrix notation:

$$F_d(\mathbf{n}) = \sum_{\mathbf{n}} F_c(\mathbf{x})\delta(\mathbf{x}+\mathbf{V}\cdot\mathbf{n}) = \sum_{\mathbf{n}} F_c(\mathbf{V}\cdot\mathbf{n})\delta(\mathbf{x}+\mathbf{V}\cdot\mathbf{n}) \tag{A.5}$$

The notation has been slightly abused: the summation is in fact a triple summation, and the vector sampling pulse $\delta(\mathbf{x}+\mathbf{V}\cdot\mathbf{n})$ is the product of three one-dimensional pulses.

A.4.1. Progressive and interlaced video

For the progressive video, the matrix **V** can be expressed as:

$$\mathbf{V} = \begin{bmatrix} \Delta x & 0 & 0 \\ 0 & \Delta y & 0 \\ 0 & 0 & \Delta t \end{bmatrix} \tag{A.6a}$$

while for interlaced video the matrix **V** can be expressed as:

$$\mathbf{V} = \begin{bmatrix} \Delta x & 0 & 0 \\ 0 & 2\Delta y & \Delta y \\ 0 & 0 & \Delta t/2 \end{bmatrix} \tag{A.6b}$$

The analysis of (A.6a) is straightforward. It is also easy to understand (A.6b). In that case, the position of the sample on the y-axis is given by

$$y = m2\Delta y + k\Delta y \tag{A.7}$$

Hence the position of the sample is displaced by Δy at the discrete time $k = 1,3,5,\ldots$ (Figure 1). From figure 1 it is possible to appreciate the column vector notation for **V**. The vectors $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$ determine the position of the first samples, and so of the whole lattice. This is particularly evident for the interlaced video, since the oblique position of $\mathbf{v}_3$ determines the interlaced lattice. Note that in the latter case, we have not depicted the x-axis for simplicity. However, in interlaced format the horizontal sampling is identical to the progressive case (left side of figure 1).

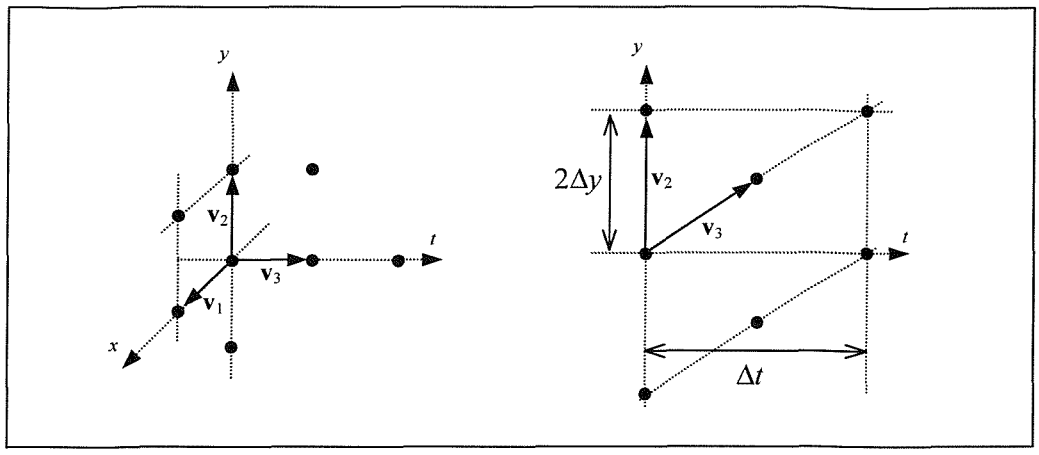

Figure 1. Sampling lattice and **V** vectors. Progressive (left) and interlaced (right) video.

A.5. Vector form of the Fourier transform

It is possible to express the Fourier transform in a vector notation applying a formula like (A.4b) to the frequency components. Define a frequency vector

$$\mathbf{f} = [f_x \quad f_y \quad f_t]^T \tag{A.8}$$

the Fourier transform of the video signal $F_c(\mathbf{x})$ can be expressed as

$$\Phi_c(\mathbf{f}) = \int_{-\infty}^{+\infty} F_c(\mathbf{x}) exp\left(-2\pi i \mathbf{f}^T \mathbf{x}\right) d\mathbf{x} \tag{A.9a}$$

and the inverse transform will be expressed as:

$$F_c(\mathbf{x}) = \int_{-\infty}^{+\infty} \Phi_c(\mathbf{f}) exp\left(2\pi i \mathbf{f}^T \mathbf{x}\right) d\mathbf{f} \tag{A.9b}$$

Note the integrals in (A.9a) and (A.9b) are triple integrals.

We can obtain the vector Fourier transform for the sampled signal:

$$\Phi_d(\mathbf{f}) = \sum_{-\infty}^{+\infty} F_c(\mathbf{V}\mathbf{n}) exp\left(-2\pi i \mathbf{f}^T \mathbf{V}\mathbf{n}\right) \tag{A.10a}$$

and the inverse transform will be expressed as:

$$F_c(\mathbf{V}\mathbf{n}) = \int_{-1/2}^{+1/2} \Phi_d(\mathbf{f}) exp\left(2\pi i \mathbf{f}^T \mathbf{V}\mathbf{n}\right) d\mathbf{f} \tag{A.10b}$$

However, substituting (A.4b) directly into (A.9b) leads to

$$F_d(\mathbf{V}\mathbf{n}) = \int_{-\infty}^{+\infty} \Phi_c(\mathbf{f}) exp\left(2\pi i \mathbf{f}^T \mathbf{V}\mathbf{n}\right) d\mathbf{f} \tag{A.11}$$

Making the change of variables $\mathbf{v} = \mathbf{V}^T\mathbf{f}$, with $d\mathbf{v} = |\mathbf{V}| d\mathbf{f}$ and $\mathbf{f} = \mathbf{U}\mathbf{v}$, where $\mathbf{U} = \left(\mathbf{V}^T\right)^{-1}$, gives

$$F_d(\mathbf{V}\mathbf{n}) = \int_{-\infty}^{+\infty} |\mathbf{V}|^{-1} \Phi_c(\mathbf{U}\mathbf{v}) exp\left(2\pi i \mathbf{v}^T \mathbf{n}\right) d\mathbf{v} \tag{A.12}$$

Dividing the integration over the volume $\mathbf{v}$ into intervals between $-1/2$ and $1/2$:

$$F_d(\mathbf{V}\mathbf{n}) = \int_{-1/2}^{+1/2} \sum_{\mathbf{k}} |\mathbf{V}|^{-1} \Phi_c[\mathbf{U}(\mathbf{v}-\mathbf{k})] exp\left(2\pi i \mathbf{v}^T \mathbf{n}\right) exp\left(-2\pi i \mathbf{k}^T \mathbf{n}\right) d\mathbf{v} \tag{A.13}$$

Comparing (A.13) with (A.10b), and considering that $exp\left(-2\pi i \mathbf{k}^T \mathbf{n}\right) = 1$ ($\mathbf{k}$ and $\mathbf{n}$ are both integers), one finally obtains

$$\Phi_d(\mathbf{f}) = |\mathbf{V}|^{-1} \sum_{\mathbf{k}} \Phi_c(\mathbf{f} - \mathbf{U}\mathbf{k}) \qquad \text{(A.14)}$$

which expresses the spectrum of the sampled signal in terms of replicas of the continuous spectrum. The periodicity of the sampled spectrum will be determined by the matrix **U**, hence by the sampling matrix **V**.

A.5.1 Progressive and interlaced video

In both cases it is easy to calculate **U** from (A.6a) and (A.6b).

$$\mathbf{U} = \begin{bmatrix} \Delta x^{-1} & 0 & 0 \\ 0 & \Delta y^{-1} & 0 \\ 0 & 0 & \Delta t^{-1} \end{bmatrix} \qquad \text{(progressive)} \qquad \text{(A.15a)}$$

$$\mathbf{U} = \begin{bmatrix} \Delta x^{-1} & 0 & 0 \\ 0 & (2\Delta y)^{-1} & 0 \\ 0 & \Delta t^{-1} & (\Delta t/2)^{-1} \end{bmatrix} \qquad \text{(interlaced)} \qquad \text{(A.15b)}$$

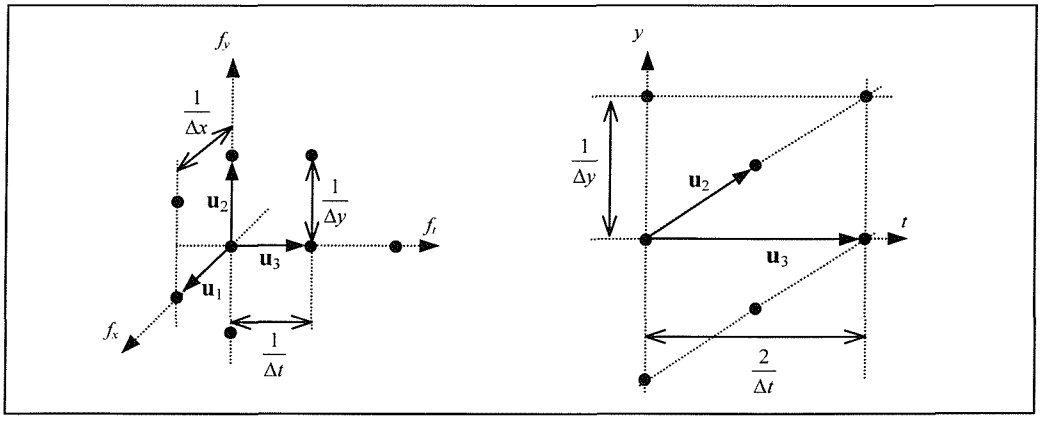

Figure 2. Spectrum and **U** vectors. Progressive (left) and interlaced (right) video. For simplicity, we depict only the centroids (black dots) of the replicas.

Again, we can express the matrix **U** in terms of column vectors $\mathbf{u}_1$, $\mathbf{u}_2$ and $\mathbf{u}_3$, that will determine the periodicity of the spectrum (figure 2). The figures in chapter 1 are drawn according to the results obtained in this section. A more general discussion on the subject can be found in Tekalp (1995). Dubois (1985) has extended the presented results to an arbitrary number of dimensions.

B. COST CONSIDERATIONS

B.1. Introduction

A meaningful comparison between the performances of RBFN and Volterra de-interlacers can not be conducted solely on the basis of the MSE. In fact, the universal approximation principle (Chapter 2) indicates that RBFN can always perform better than any given Volterra series, provided that the number of centres is large enough. Some account of the computational complexity should be taken to make a realistic comparison.

The main field of application for de-interlacing techniques is digital video processing. Real-time or near real-time applications often require the algorithms to be built in hardware, most likely designed at port-level using programmable gate arrays (PGA).

Computational complexity, in terms of the number of elementary computational units, is a measure of cost of paramount importance at the hardware level. Another parameter that influences the cost is the degree of parallelism of the algorithm, since this influences the processing time. Note that both Volterra series and RBFN have been defined as single-layer networks. However, this definition is not correct in terms of elementary computational units, since the relative kernels are composed of a series of simpler operations, like multiplies and additions. In this work we will not consider directly the timing problems. Our analysis will hence focus on the more general computational complexity of the two networks.

We will basically decompose the two networks in terms of scalar multipliers and scalar adders, and the cost will be expressed as a function of their respective costs, C_{mult} and C_{sum}. In the Gaussian RBFN we have to consider a third component, the scalar exponential and its cost C_{exp}. In practice it is difficult to determine the real costs C_{mult}, C_{sum} and C_{exp}. A complete analysis is beyond the scope of this work. Hence we will use a mixture of practical and qualitative considerations to estimate some form of relative cost of the networks. Specifically, we will make the reasonable

assumption that the scalar adder has the lowest cost, and consider the results for a sensible range of possible costs C_{mult} and C_{exp}, relative to C_{sum}.

B.2. Computational cost of the 3rd-order Volterra series

The number M of branches in the network diagram of the Volterra series is not a free parameter as in RBFN, but is determined by the degree p of the series and the dimension D of the input space. For a 3rd order series, we can identify 3 groups of branches, with D, $\frac{D(D+1)}{2}$ and $\frac{D\left(D^2+3D+2\right)}{6}$ elements, corresponding to the 3 degrees of the expansion (Figure 1), and there is a different cost per branch for each group. The cost of the 3rd order branches can be reduced by pipe-lining the 3rd order group with the output of 2nd order branches. (Figure 1). The linear group is considered to be of cost zero, since it merely transfers the input in the intermediate space $\mathbf{h}$.

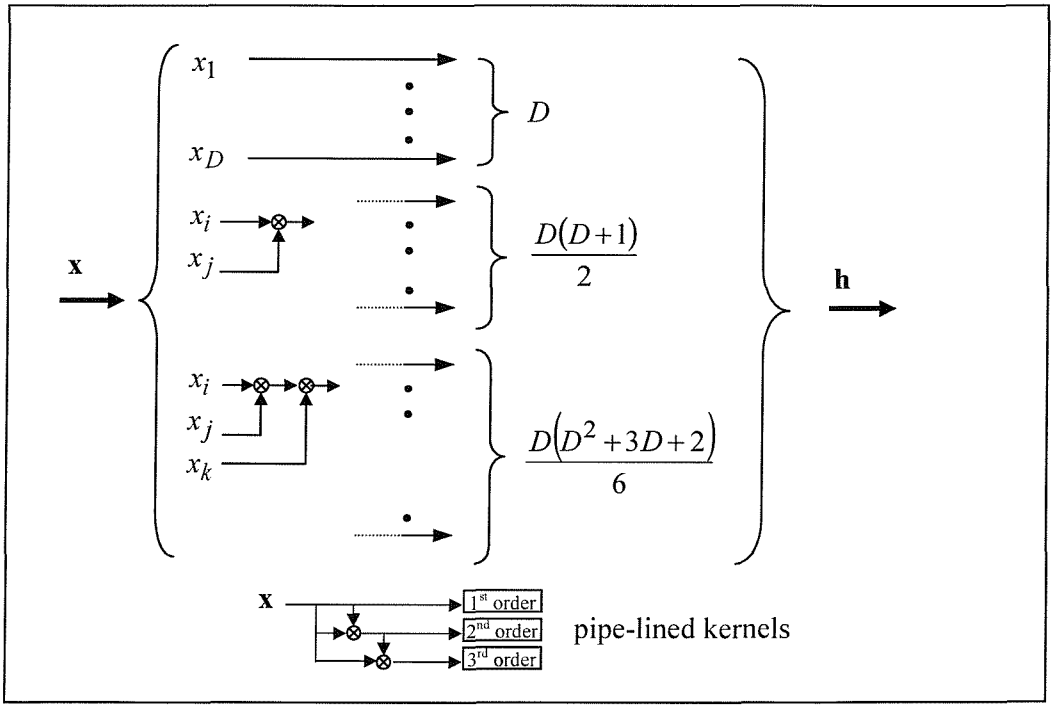

Figure 1. Branch decomposition of the 3rd-order Volterra network

The cost per branch for each group is given by:

$$C_{(1)} = 0$$
$$C_{(2)} = \frac{D(D+1)}{2} \cdot C_{mult} \qquad \text{(A.1)}$$
$$C_{(3)} = \frac{D(D^2+3D+2)}{6} \cdot C_{mult}$$

Note that we have used the pipe-lined structure to compute the cost of the 3rd-degree group. To complete the cost evaluation we have to consider the number of output multipliers (the weights of the series) $D + \frac{D(D+1)}{2} + \frac{D(D^2+3D+2)}{6}$, and the number of output summations, $D + \frac{D(D+1)}{2} + \frac{D(D^2+3D+2)}{6} - 1$.

The computational cost of a 3rd order Volterra series, with $D = 8$, is then:

$$C_{VOLT} = 320 \cdot C_{mult} + 164 \cdot C_{sum} \qquad \text{(A.2)}$$

B.3. Computational cost of Radial Basis Function Networks

The analysis of the cost of the RBFN is more complicate than the Volterra series. Here we have a kernel that comprises of vector and scalar multipliers and adders, and we have to evaluate the cost of implementing the Gaussian function. The vector elements can be easily expressed in terms of scalar elements, once we determine a value for D.

The evaluation of C_{exp} is more complicate. Basically, the most immediate way to realise an arbitrary scalar-to-scalar function in hardware is to use a look-up table (LUT). Since the LUT is basically a read-only memory (ROM), its performance is limited by the size of its memory, i.e. the bit-resolution of the dynamics involved. On the other hand, the computational effort can be considered very small. Clearly the cost function depends mainly on the size of the LUT. The memory size influences the

coarseness of the approximation of the LUT to the non-linear function chosen, and hence affects the MSE performance of the interpolator. The effects of such approximations have not been considered here, so a detailed cost analysis of these elements is not available.

Figure 2 shows the branch decomposition of the Gaussian RBFN. Note how the left side of the branch is kept in vector form. In fact, the scalar multiplier/scalar adder representation is more complex.

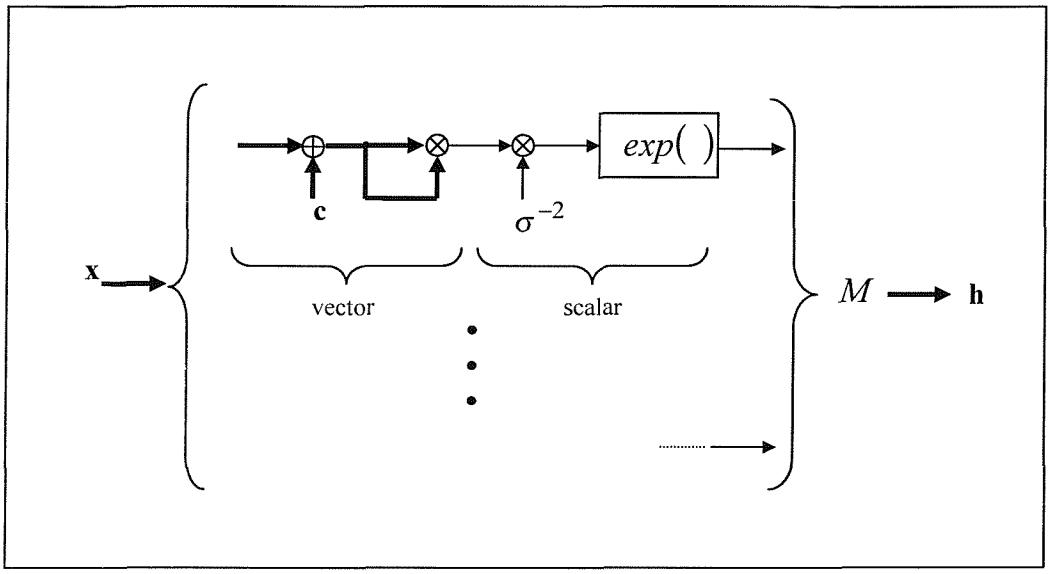

Figure 2 Branch decomposition of the Gaussian RBFN

This time the number of branches M is a free parameter of the network. The resulting cost can be easily calculated as:

$$C_{RBF} = M \cdot [D \cdot C_{sum} + (D+2) \cdot C_{mult} + C_{exp}] + (M-1) \cdot C_{sum} \qquad \text{(A.3)}$$

The cost of the hybrid architecture is equally simple to calculate:

$$C_{HRBF} = M \cdot [D \cdot C_{sum} + (D+2) \cdot C_{mult} + C_{exp}] + D \cdot C_{mult} + \\ + (M+D-2) \cdot C_{sum} \qquad \text{(A.4)}$$

B.4. Cost comparison of Volterra series and hybrid Gaussian RBFN.

The cost comparison of the two techniques is performed assuming a range 1~10 for the costs C_{mult} and C_{exp} relative to C_{sum}. We consider a set of ten hybrid RBFN networks, with centres increasing in number from 10 to 100. More detail is given to the range 10~50, since this will prove to be the range of interest to us. Figure 3 shows the cost plots against the relative value of C_{mult}, with C_{exp}=1. Figure 4 shows the same plots, with C_{exp}=10.

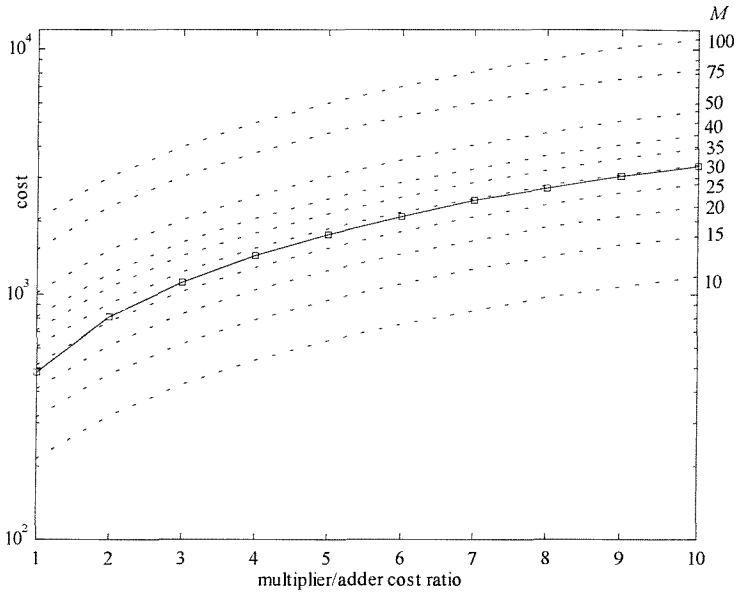

Figure 3. Volterra cost function (—) against the relative cost ratio C_{mult} / $Csum$, compared with hybrid networks (--) in the range M=10~100, C_{exp} = 1.

From the plots it seems reasonable to deduce that RBFN can be competitive compared to Volterra series if the number of centres is in the range 10~30. The lower bound is given by the assumption that for M <10, the MSE performance degrades very quickly. Hence we should pursue a training strategy to reduce the number of centres down to this range, at the same time keeping the MSE at least equal to the 3rd-order Volterra series.

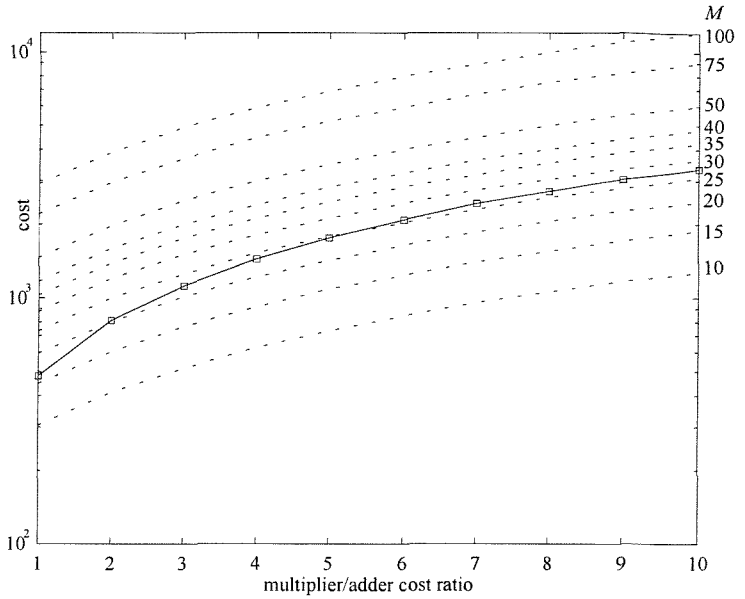

Figure 4. Volterra cost function (—) against the relative cost ratio C_{mult} / $Csum$, compared with hybrid networks (--) in the range M=10~100, C_{exp} = 10.

The results shown give an estimate for the useful range of centres that looks reasonably robust to a broad set of conditions. However, the cost model is based on very general assumptions that greatly simplify the issues involved in a real hardware implementation of the algorithms. This cost analysis is not exhaustive but nevertheless represents a reasonable framework to set our strategy in pursuing a convenient RBFN implementation of a de-interlacing system.

C. NELDER-MEAD SIMPLEX ALGORITHM

C.1. The Simplex method

The Simplex method (Spendley *et al.*, 1962) is a simple yet powerful iterative method to find minima in a function by simply observing its values (and not its derivatives). A regular simplex in a D-dimensional space is a set of equidistant D+1 points $\mathbf{x}_1 \ldots \mathbf{x}_{D+1}$, forming a D-dimensional tetrahedron (figure 1). The simplex method computes the function values at these points, $y_1 \ldots y_{D+1}$, and applies the reflection rule (figure 1) according to these values.

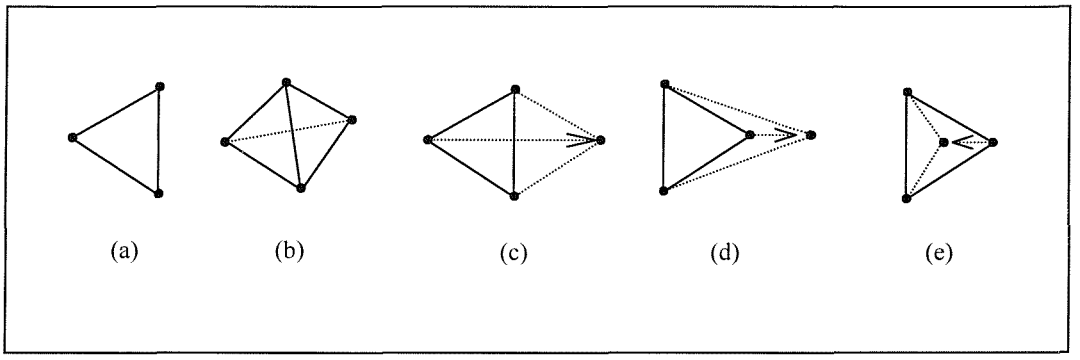

Figure 1. (a) 2-D simplex (b) 3-D simplex (c) reflection (d) expansion (e) contraction

On the first step of the iteration, the algorithm determines the maximum value y_M. The corresponding vertex $\mathbf{x}_M$ is then reflected into the centroid $\bar{\mathbf{x}}$ of the other D points, thus forming a new simplex with the old points (excluding $\mathbf{x}_M$) and the new point $\mathbf{x}^* = 2\bar{\mathbf{x}} - \mathbf{x}_M$. The function value $y^* = y(\mathbf{x}^*)$ is evaluated and the process is repeated.

After the first iteration, it is possible that the new vertex yields the largest value in the new simplex, and consequently its reflection would cause an oscillation. In which case the second largest value is selected. However, after a number of iterations the algorithm will fail to proceed, since any choice will reflect a vertex already considered. In that case, after a certain vertex $\mathbf{x}'$ has been in the current simplex for more than a fixed number of iterations I, then the simplex is reduced by replacing the

other vertices by new ones half way along the edge to the vertex $\mathbf{x}'$. Spendley *et al.* suggest the empirical relationship

$$I = 1.65D + 0.05D^2 \qquad \text{(C.1)}$$

An example for D=2 is shown in figure 2. As one can see, vertex 7 in the simplex (4,6,7) is not reflected immediately, although yielding the highest value, since it is the newest vertex and its reflection would cause an oscillation. When the simplex (6,9,10) is reached, the vertex 6 has been in the current simplex for 4 consecutive iterations, and since I=3.5, the simplex is reduced.

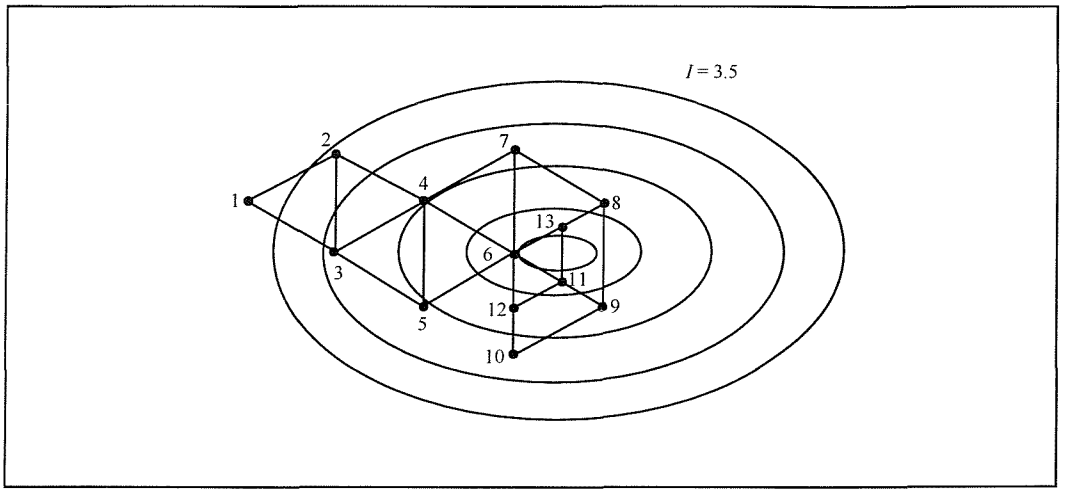

Figure 2. Simplex algorithm, convergence over a convex function in 2 dimensions

C.2. The Nelder-Mead algorithm

Nelder and Mead (1965) propose a modified simplex method that allows distortion in the simplex, in order to accommodate the local geometry of the function. Specifically, if the new vertex value y^* is smaller than $y_1 \ldots y_{D+1}$, then the new vertex is expanded (figure 1):

$$\mathbf{x}^{**} = \alpha \mathbf{x}^{*} + (1-\alpha)\bar{\mathbf{x}} \qquad \text{(C.2)}$$

where α is an expansion coefficient, and is greater than the unity. If however $y^{**} > min\{y_1 \ldots y_{D+1}\}$, then the expansion failed and $\mathbf{x}^*$ will remain the new vertex.

A failed expansion might happen for instance if the simplex reaches a steep valley, at an angle perpendicular to the valley's direction (figure 3).

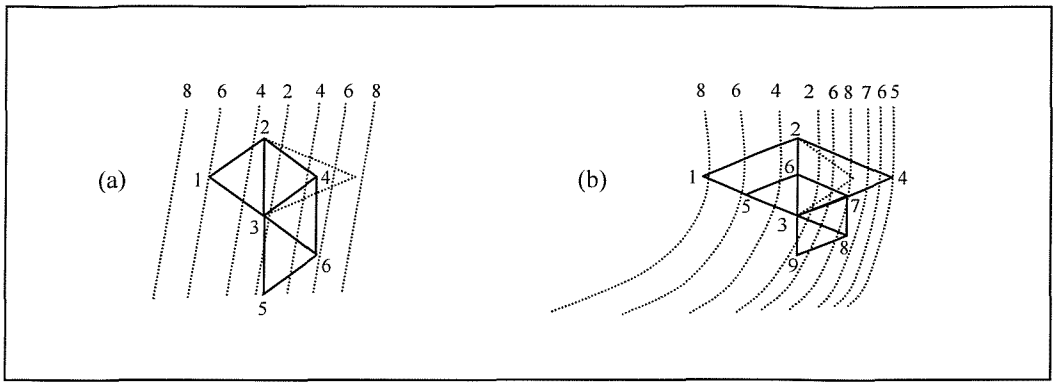

Figure 3. (a) failed expansion, (b) failed contraction. The numbers on the dashed lines represent the function's value. In (a), the simplex (2,3,4) coming from the reflection of (1,2,3) tries to expand, but it fails to do so as vertex 4 is on the rising edge of a valley. In (b), vertex 4 falls in a shallow valley contiguous to a deeper valley. Therefore, the simplex (1,2,3) is halved in the direction of vertex 3 (minimum of the simplex).

Another amendment to Spendley's algorithm is the introduction of a contraction (figure 1). If the new vertex $\mathbf{x}^*$ is such that $\underset{i \neq M}{y_i} < y^* < y_M$, so that the new vertex value is still the maximum, then $\mathbf{x}^*$ is replaced by:

$$\mathbf{x}^{**} = \beta \mathbf{x}^* + (1-\beta)\bar{\mathbf{x}} \qquad \text{(C.2)}$$

where β is a contraction coefficient, $0 \leq \beta \leq 1$. If $y^{**} < min\{y^*, y_M\}$, then $\mathbf{x}^{**}$ is the new vertex. If the latter condition is not satisfied, so that $\mathbf{x}^{**}$ produces a worse result than $\mathbf{x}^*$ and $\mathbf{x}_M$, then the contraction failed. A failed contraction is more unlikely to happen than a failed expansion (Nelder, Mead 1965). It may happen when the simplex is in a valley that curves, and one of the vertexes is farther from the valley bottom than the others. Then $\mathbf{x}^*$ could fall into a descending area, and the contraction would be ascending (figure 3). In this case, the size of the non-reflected simplex is halved in the direction of the minimum of the simplex, as in section C.1.

The final issue concerns the criterion for stopping the algorithm. One method computes the standard deviation of the values

$$e = \sqrt{\sum_{i=1}^{D+1} (y_i - \bar{y})^2} \qquad \text{(C.3)}$$

If (C.3) falls below a given value, the algorithm stops.

C.3. Estimate of the Hessian matrix

The minimisation methods proposed in the previous sections are independent of the analytic properties of the function, specifically the second derivatives expressed by the Hessian matrix $\mathbf{H} : h_{ij} = \partial y^2 / \partial x_i \partial x_j$, and the gradient vector $\mathbf{g} : g_i = \partial y / \partial x_i$. However, it is desirable to have an estimate of these two quantities, since almost any minimisation problem can be approximated by a quadratic function in a sufficient small neighbourhood of the minimum (Fletcher, 1987), and therefore can benefit from the analytic solution of the quadratic problem.

Nelder and Mead (1965) provide a method to estimate the Hessian matrix and the gradient vector from the vertices of the simplex. We will not describe the details of the derivation. They point out two major hazards of their proposed estimate. The first is that, if the simplex is too small, the differences in the vertex values might consist largely of rounding errors, therefore producing poor estimates. The second hazard might occur if the simplex is too large, and the estimates are again poor. The authors suggest that, provided a convergence has occurred, the former rather than the latter is the main risk to avoid. In which case the solution proposed is to enlarge the simplex, so that the effect of rounding errors is mitigated.

BIBLIOGRAPHY

Akaike H.
Information Theory and an Extension of the Maximum Likelihood Principle
In B. N. Petrov and F. Csaki (Eds.)
2nd International Symposium on Information Theory
pp. 267-281. Tsahkadsov, Armenia, USSR 1973

Arad N., D. Reisfeld
Image Warping using few Anchor Points and Radial Functions
Computer Graphics Forum
volume 14, pages 35-46. Eurographics, Basil Blackwell Ltd, 1995. Eurographics '95 Conference issue.

Barron A.R.
Universal Approximation Bounds for Superposition of a Sigmoidal Function
IEEE transactions on information theory
Vol. 39, No 3, May 1993

Bellers, E.B.
De-interlacing. A Contribution to the Interlaced versus Progressive Video Debate
Proefschrift Technische Universiteit Delft – Philips Electronics N.V. 1999
CIP-Gegevens Koninklijke Bibliotheek, Den Hag

Bellman, R.
Adaptive Control Processes: A Guided Tour
Princeton University Press, 1961

Bertero M., T. Poggio, V. Torre
ill-Posed Problems in Early Vision
Proceedings of the IEEE
76: 869-889, 1988

Bishop C. M.
Curvature Driven Smoothing: a Learning Algorithm for Feedforward Networks
IEEE transactions on neural networks
4 (5) , 882-884 1993

Bishop C. M.
Novelty Detection and Neural Network Validation
IEE proceedings: vision, Image and Signal Processing
141 (4), 217-222 special issue on applications of neural networks, 1994

Bishop, C.M.
Neural Networks for Pattern Recognition
1995 Oxford University press

Björk, A.
Solving Linear Least Squares Problems by Gram-Schmidt Orthogonalisation
Nordisk Tidshrift for Informationsbehadlung
7, 1-21, 1967

Bridle J.S.
Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationship to Statistical Pattern Recognition
In F. Fogelman Soulie' and J. Herault (Eds.)
Neurocomputing: Algorithms, Architectures and Applications
pp.227-236, New York, Springer-Verlag 1990

Broomhead D.S., D. Lowe
Multivariable Functional Interpolation and Adaptive Networks
complex systems,
2 pp.321-355, 1988

Cha I., Kassaam A.
RBFN Restoration of Nonlinearly Degraded Images
IEEE transactions on image processing
Vol. 5, No. 6, June 1996

Chen S., S.A. Billings, W. Luo
Orthogonal Least Squares Methods and their Application to Non-Linear System Identification
Int. J. Control
Vol. 50, No. 5, pp. 1873-1896, 1989

Chen S., S.A. Billings, C.F. Cowan, P.M. Grant
Practical Identification of NARMAX Models Using Radial Basis Functions
Int. J. Control
Vol. 52, No. 6, pp. 1327-1350, 1990

Chen S, C.F.N. Cowan, P.M. Grant
Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks
IEEE transactions on neural networks
Vol. 2, pp. 302-309, No. 2,March 1991

Coley D. A.
An introduction to genetic algorithms for scientists and engineers
Singapore : World Scientific, 1999

Collis W.B.
Higher-order Spectra and their application to nonlinear mechanical systems
PhD Thesis
Institute of Sound and Vibration Research, University of Southampton, UK, 1996

Collis W.B., M. Weston, and P.R. White
The Application of Non-linear Volterra Type Filters to Television Images
IEEE workshop on Higher Order Spectra, Banff
Canada, pp. 1-10, 1997.

Dempster A. P., N. M. Laird, D. B. Rubin
Maximum Likelihood from Incomplete Data via EM Algorithm
J. Royal Statist. Soc.
Ser. B (Methodological)
1997, No. 39, pp. 1-38

De Stefano A., P.R. White, W.B. Collis
An Innovative Approach for Spatial Video Noise Reduction Using a Wavelet Based Frequency Decomposition
Proceedings of ICIP2000 conference
September 2000

Domingos P.
The role of Occam's razor in knowledge discovery
Data Mining and Knowledge Discovery
3(4): 409-425, 1999

Drucker H., C. J. C. Burges, L. Kaufman, A. Smola, V. Vapnik
Support Vector Regression Machines
In M. Mozer, M. Jordan, and T. Petsche, editors
Advances in Neural Information Processing Systems
9, pages 155-161, Cambridge, MA, 1997. MIT Press.

Dubois, E.
The Sampling and Reconstruction of Time Varying Imagery with Application in Video Systems
Proceedings IEEE
Vol. 73, No 4, pp. 502-522, April 1985

Duda, R. O., P.E. Hart
Pattern Classification and Scene Analysis
New York: John Wiley, 1973

Fletcher, R.
Practical Methods of Optimisation
1987 Second Edition Wiley and Sons

Fung, C. F., S. A. Billings, W. Luo
Online Supervised Adaptive Training Using Radial Basis Function Networks
Neural Networks
vol. 9, pp. 1597-1617, 1996

Geman, S., E. Bienestock, and R. Doursat
Neural Networks and the Bias/Variance Dilemma
Neural Computation
1992, 4 (1), 1-58

Giordano A., F. M. Hsu
Least Squares Estimation with Applications to Digital Signal Processing
John Wiley & Sons 1985

Girosi F., T. Poggio
A Theory of Networks for Approximation and Learning
A.I.Memo No.1140, C.B.I.P. paper No.31
MIT Artificial Intelligence Laboratory July 1989

Girosi F., T. Poggio (b)
Networks and the Best Approximation Property
AIM-1164, 22, 1989

Girosi F., T. Poggio,
Networks for Approximation and Learning
Proceedings of the IEEE,
Vol. 78 No. 9, September 1990

Girosi F., M. Jones, T. Poggio
Regularization Theory and Neural Networks Architectures
Neural Computation
vol. 7 No 2 219-269, 1995

Goldberg, D E., 1953
Genetic algorithms in search, optimization, and machine learning
Reading, Mass : Addison-Wesley, 1989

Golub G.H., C. Reinsch
Handbook Series Linear Algebra: Singular Value Decomposition and Least Squares Solutions
Numer. Math.
14, pp. 403-420, 1970

Gutta S, Wechsler H.
Face Recognition using Hybrid Classifier Systems.
Proceedings of the IEEE International Conference on Neural Networks,
1996. p.1017-1022.

Hartman E. J., J. D. Keeler, J. M. Kowalski
Layered Neural Networks with Gaussian Hidden Units as Universal Approximations
Neural Computation
1990, 2 (2), 210-215

Hassibi B., D.G. Stork
Second Order Derivatives for Network Pruning: Optimal Brain Surgeon
In S.J. Hanson, J.D. Cowan, C.L. Giles (Eds)
Advances in Neural Information Processing Systems
Vol. 5, pp. 164-171, San Mateo CA: Morgan Kaufmann 1993

Hinton G.E,
Learning Translation Invariant Recognition in Massively Parallel Networks
In J. W. de Bakker, A. J. Nijman, and P. C. Treleaven (Eds.)
Proceedings PARLE Conference on Parallel Architectures and Language Europe
pp. 1-13, Berlin; Springer - Verlag.

Howell, A. J., H. Buxton
Invariance in Radial Basis Function Neural Networks in Human Face Classification
Technical Report CSRP 365, School of Cognitive and Computing Sciences, University of Sussex. 1995

Jacobs R. A. M., I. Jordan, S. J. Nowlan, G. E. Hinton
Adaptive Mixtures of Local Experts
Neural Computation
3 (1) 1991, pp 79-87

Jones R.D.
Function Approximation and Time Series Prediction With Neural Networks
Proc. Int.Joint Conf. Neural Networks
Vol. 4, 1991, pp 89-102

Jung Y., B. Choi, Y. Park, S. Ko,
An Effective De-interlacing Technique using Motion Compensated Interpolation
IEEE Transactions on Consumer Electronics,
Vol. 46, N0 3, August 2000

Kassaam S.A., I. Cha
Radial Basis Function Network in Non-Linear Signal Processing Applications
Proc. 27th Annual Asilomar Conf. Signals, Syst., Comput.
Asilomar CA, Nov. 1993, Vol. 1, p. 1022-1025

Kohonen T.
The Self Organising Map
Proceedings of the IEEE
vol. 78, No. 9, september 1990

Kolmogorov A. N.
On the representation of Continuous Functions of Several Variables by Superposition of Continuous Functions of One Variable and Addition
Dokl. Akad.Nauk SSSR
114:953-956, 1957

Le Cun Y., J.S. Denker, S.A. Solla
Optimal Brain Damage.
In D.s. Touretzky (Ed.),
Advances in Neural Information Processing Systems
Volume 2, pp. 598-605. San Mateo CA: Morgan Kaufmann 1990

Lorentz G. G.
On the 13th Problem of Hilbert
Proceedings of Symposia in Pure Mathematics
pages 419-429, Providence, RI, 1976. American Mathematical Society

McLachlan G. J., K. Basford
Mixture Models: Inference and Applications to Clustering
1988 New York: Marcel Dekker

Micchelli C.A.
Interpolation of Scattered Data: Distance Matrices and Conditionally Positive Definite Functions
Constr. Approx. 1986, 2: 11-22
Mitchell M.
An Introduction to Genetic Algorithms
Cambridge, Mass.: MIT, 1998

Moody J., C. J. Darken
Fast Learning in Networks of Locally-Tuned Processing Units
Neural Computation
1 (2), pp. 281-294, 1989

Morozov V.A.
Methods for Solving Incorrectly Posed Problems
Springer-Verlag, Berlin 1984

Musavi M.T., W. Ahmed, K. H. Chan, K. B. Faris, D.M. Hummels
On The Training of Radial Basis Function Classifiers
Neural Networks
Vol. 5, pp. 595-603, 1992

Nelder J.A., R. Mead
A Simplex Method for Function Minimisation
Computer journal
Vol. 7 pp. 308-313, 1965

Nikias L.N., A. P. Petropulu
Higher-order Spectra analysis
Prentice Hall signal processing series
Prentice Hall, New Jersey, 1993

Papoulis A.
Probability Random Variables and Stochastic Processes
Mcgraw-Hill, New York 1984

Penrose R.
A Generalized Inverse for Matrices
Proceedings of the Cambridge Phil. Soc.
51, 406-413, 1955.

Perlovsky L.I., M.M. McManus
Maximum likelihood Neural Networks for Sensor Fusion and Adaptive Classification
Neural Networks
Vol.4, pp. 89-102, 1991

Perrone M. P.
General Averaging Results for Convex Optimisation
Proceedings 1993 Connectionist Models Summer School
M.C. Mozer et al. (Eds.) pp. 364-371. Hillsdale, NJ: Lawrence Erlbaum, 1993

Perrone M. P., L. N. Cooper
When Networks disagree: Ensemble Methods for Hybrid Neural Networks
Artificial Neural Networks for Speech and Vision
R.J. Mammone (Ed.) pp. 126-142. London: Chapman & Hall (1993)

Petrou M., P. Bosdogianni
Image Processing – the Fundamentals
John Wiley and Sons, LTD 1999

Powell M. J. D.
Radial Basis Function Approximations to Polynomials
Proceedings of the 12th Biennal Numerical Analysis Conference
Dundee, UK pp. 223-241, 1987

Oppenheim A. V., R.W. Schafer
Digital Signal Processing
Prentice-Hall 1974

Redner R.A., H.F. Walker
Mixture Densities, Maximum Likelihood and the EM Algorithm
SIAM review
Vol. 26, No. 2, April 1994

Rice J.R.
Experiments on Gram-Schmidt Orthogonalisation
Mathematics of Computation
20, 325-328, 1966

Rosenblum M., Y. Yacoob and L.S. Davis
Human Emotion Recognition from Motion Using a Radial Basis Function Network Architecture
IEEE Workshop on Motion of Non-Rigid and Articulated Objects
Austin, 1994

Rumelhart D. E., G. E. Hinton, R. J. Williams
Learning Internal Representations by Error Propagation
In D. E. Rumelhart, J. L. McClelland, and the PDP Research Group (Eds.)
Parallel Distributed Processing: Explorations in the Microstructure of Cognition
Volume 1: Foundations, pp. 318-362. Cambridge, MA: MIT press 1986

Schetzen M.
The Volterra and Wiener Theory of Non Linear Systems
Krieger Publishing company, 1980 Malabar, Florida

Schetzen M.
Nonlinear System Modeling Based on the Wiener Theory
Proceedings of the IEEE,
Vol. 69, No 12, December 1981

Schoenberg I. J.
Metric Spaces and Completely Monotone Functions
Ann. of math., 44:522-841, 1938

Sherstinsky A., R. W. Picard
On Training Gaussian Radial Basis Functions for Image Coding
Tech. Rep. 188, M.I.T. Media Lab Vision and Modeling Group, Feb. 1992.

Smola A. J.
Regression Estimation with Support Vector Learning Machines.
Master's thesis, Technische Universität, München, 1996.

Smola A.J., B. Scholkopf
A Tutorial on Support Vector Regression
NeuroCOLT2 Technical Report Series
Produced as part of the ESPRIT Working Group in Neural Computation and Learning II
NC2-1998-030 October 1998
Spendley W., G. R. Hext, F. R. Himsworth
Sequencial Application of Simplex Designs in Optimisation and Evolutionary Operation
Technometrics,
Vol. 4, No. p. 441, 1962

Streit R.L., T.E.Luginbuhl
Maximum Likelihood Training of Probabilistic Neural Networks
IEEE transactions on neural networks
Vol. 5, No. 5, September 1994

Stroebel L., R. Zakia
The Focal Encyclopedia of Photography
Focal Press, 1993

Sugiyama K., H. Nakamura
A Method of De-interlacing with Motion Compensated Interpolation
IEEE transactions on Consumer Electronics
Vol. 45, No.3, August 1999

Tekalp, M. A.
Digital Video Processing
Prentice Hall PTR 1995

Tikhonov, A.N.
Solution of Incorrectly Formulated Problems and the Regularisation Method
Soviet Math. Dokl.
4: 1035-1038, 1963

Tikhonov, A.N., V. Y. Arsenin
Solution of ill-posed problems
Washington D.C.: V. H. Winston 1977

Titterington D. M., A. F. M. Smith, U.E. Makov
Statistical analysis of Finite Mixture Distributions
New York, John Wiley 1985

Tompa D., J. Morton, E. Jernigan
Perceptually Based Image Comparison
Proceedings of ICIP2000 conference
September 2000

Van Trees H. L.
Detection, Estimation, and Modulation Theory
John Wiley and Sons, 1971

Vapnik V., A. Lerner
Pattern Recognition Using Generalised Portrait Method
Automation and remote control
24, 1963

Vapnik V., A. Chevronenkis
A note on one class of perceptrons
Automation and remote control
25, 1964

Vapnik V., S. Golowich, A. Smola
Support vector method for function approximation, regression estimation and signal processing
In M. Mozer, M. Jordan, T. Petsche, editors
Advances in neural Information Processing Systems 9
Pages 281-287, Cambridge, MA, MIT Press 1997.

Vitushkin A. G., G. M. Henkin
Linear Superposition of Functions.
Russian Mathematical Surveys
22:77-125, 1967

Volterra, V.
Theory of Functionals and of Integral and Integro-Differencial Equations
New York: Dover, 1959.
Weiss Y.
Smoothness in Layers: Motion Segmentation using Non-parametric Mixture Estimation.
CVPR, pp. 520--526, 1997

Yuille A., N. M. Grzywacz
The Motion Coherence Theory
Proceedings of the International Conference on Computer Vision
pages 344-354, Washington D.C. , December 1988 IEEE Computer Society Press