School of Computing

Model-Based Neural Networks For Invariant Pattern Recognition

David McGregor Squire

This thesis is presented as part of the requirements for
the award of the Degree of Doctor of Philosophy
of the
Curtin University of Technology

October 27, 1996

Acknowledgments

All attempts to acknowledge those who have helped a large project to completion are prone to the risk that someone may be left out. Nevertheless, I commence by thanking my supervisor, Terry Caelli. I cannot imagine a PhD. student being more fortunate than to have Terry as a supervisor. Throughout the entire process he has remained enthusiastic, encouraging and stimulating. Moreover, he has been a friend, providing support of all kinds.

Special thanks must go to my two good friends Tom Wild and Michael Merrylees. It is due to them that I commenced working towards this thesis. Tom introduced me to Terry Caelli, and we shared an introduction to neural networks programming together. There are functions in the **Xnet** package for which he is responsible. Michael was responsible for convincing me that taking on this project was a good idea, and has always been a fount of good cheer and good ideas. My other close friends, though perhaps not directly involved academically, have been vital in sustaining me. They know who they are.

All the members of the Vision Group, in both its Melbourne University and Curtin University incarnations, have collaborated to produce a happy, stimulating environment. In particular, I would like to thank Craig Dillon and Mark Ollila for help at various times with C programming and typesetting with LaTeX. John Krivitsky has provided similar help, and has also responded promptly and effectively to my system administration requests, which were no doubt less than reasonable at times. Tom Drummond was a marvelous universal sounding-board, particularly in mathematical matters. All of them have become friends.

Finally, I must thank my parents, who have been constantly supportive throughout the years I have worked towards this thesis, as indeed they have always been. They have always encouraged me, and fostered self-belief with their faith and love.

David Squire
February 10, 1997

Abstract

In this thesis, the notion of *Model-Based Neural Networks* is introduced. Model-Based Neural Networks, whilst retaining the essential structure and advantages of traditional neural networks, enable explicit modeling of the target system. This can result in dramatically improved generalization of classification performance to patterns not present in the training data, and representation in considerably lower-dimensional state spaces.

An important problem in many areas of computer vision is invariant pattern recognition. This is the chosen domain for demonstrating the efficacy of the Model-Based Neural Network approach. Model-Based Neural Networks can be constructed which have responses which are invariant to specified transformations of the input data. Such networks can be trained with much smaller training sets than are required by traditional networks, since it is no longer necessary to provide examples of transformed versions of the input prototypes. This, coupled with the reduction in the dimensionality of the parameter space, means that training such networks is often much less computationally-expensive than the traditional alternative.

To situate this work, a review of existing general techniques for invariant pattern recognition is presented in Chapter 2, and of previous neural network-based approaches in Chapter 3. Chapter 4 presents a variety of different forms of Model-Based Neural Network, and demonstrates their utility on a range of invariant pattern recognition problems, both real and synthetic. Included is a comparison with an earlier study, which reveals the great improvements possible with Model-Based Neural Networks.

Chapter 5 introduces a new invariant feature of two-dimensional contours, the Invariance Signature. It is shown in Chapter 6 that a Model-Based Neural Network can be constructed which calculates this multi-dimensional feature, and classifies patterns on this basis. Chapter 7 reports experimental results demonstrating that such Invariance Signature-based MBNNs can be employed successfully for shift-, rotation- and scale-invariant optical character recognition.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

Many claims have been made concerning the importance of neural networks (NNs) as a paradigm shift in modeling both nature and the central processes of Information Technology including, most directly, Artificial Intelligence problem domains. To this stage, NNs have been applied to many and varied areas of inquiry from the control of chemical plants, through to pattern recognition, and many biological domains. Further, most proponents do not claim that NNs literally correspond to what actually occurs in the human brain, but there is a general belief of a loose correspondence between the actual computational units used and the response properties of individual neurons. Added to this, there is a belief that the inherent parallelism of NNs is consistent with brain function and that the use of fundamental numerical computations, in contrast to symbolic or declarative representations, is representative of the "hardware" of intelligent behaviour. This thesis is not aimed at challenging these claims, but at considering fundamental computational problems of NNs related to their usual representational dimensionality and model adequacy. In particular, this thesis is concerned with the problem of integrating domain- or task-specific knowledge into the very architecture of NNs.

It is important to note that most past NN formulations have a few central features in common. They are:

1. Problems are solved by the determination of weights or states in an extremely high-dimensional state space.

2. Learning, parameter estimation, and information processing are all inherently parallel.

3. No further constraints on the system are required, since the NN learns weights which are necessary and sufficient to predict behaviour.

This indicates that most applications of NNs are based upon the assumption that solutions to problems can be obtained by using generic technologies which essentially search high-dimensional state spaces, and so require *no additional knowledge* about the system under analysis. Examples of this abound in the literature, where the input and output level responses are defined by discrete nodes and their transducer functions, and at least one hidden layer is introduced. Further, NNs usually employ Supervised Learning which, in one sense, is a form of constraining the system and weight estimation processes. However, it is usually "black-box" in so far as it, *per se*, makes no assumptions about the processing characteristics and desired properties of the system not explicit in the training samples. For example, most traditional NN approaches to pattern recognition lack explicit shift, rotation and scale invariances, as the NNs are not modeled with such characteristics in mind. If such invariances arise it is due to the presence of appropriately shifted, rotated, or scaled example patterns in the training data set.

The aim of this thesis is to formalize an emerging perspective to NNs [e.g Tebelski and Waibel, 1990; Waibel, Jain, McNair, Saito, Hauptmann and Tebelski, 1991] which, whilst retaining the essentials of the NN approach, enables explicit modeling of the target system. This can result in dramatically improved generalization of classification performance to patterns not present in the training data, and representation in considerably lower-dimensional state spaces. Perhaps most importantly, model-based NNs can be constructed so that they are guaranteed to have responses which are invariant under certain transformations of the input data. Such networks can be trained with very much smaller training sets, since it is no longer necessary to provide examples of transformed versions of the input prototypes. This, coupled with the reduction in the dimensionality of the parameter space, means that training such model-based NNs is often much less computationally-expensive than the traditional alternative.

To attain these goals we first define the classical formulation, compare it to past technologies, and then develop the model-based formulation.

1.1.1 The Classical Feed-forward Neural Network

For a classical feed-forward NN, the input x_j to a given "neuron", j, is defined by

$$x_j = \sum_i y_i w_{ij} \tag{1.1}$$

and the output is defined by the logistic function

$$y = \frac{1}{1 + e^{-x}}. \tag{1.2}$$

This process is implemented in NNs with at least three layers: an input layer, and output layer and one or more hidden layers. These are connected in the form of a feed-forward architecture, as shown in Figure 1.1 (as, for example, employed by Plaut and Hinton (1987)). In classification or recognition problems, the input layer is defined by an array of nodes which constitute a sampled version of the input signal. The output layer is defined by a set of nodes each corresponding to a class, pattern type, or category. Connections $\{w_{ij}\}$ between nodes are usually restricted to layers above and below a given layer (see Figure 1.1). For a 3-layer NN with 100, 20, and 10 nodes for the three layers, respectively, this results in $100 \times 20 + 20 \times 10 = 2200$ connections. Each such connection has a weight to be estimated in accordance with the desired input-output (I-O) characteristics (Equations 1.1 and 1.2 above).

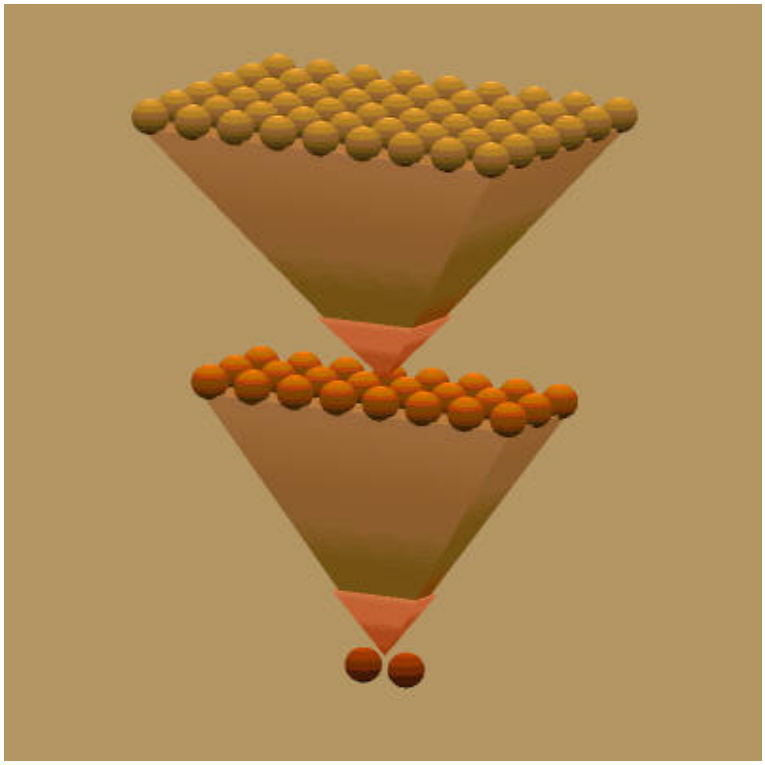


Figure 1.1: A classical feed-forward network. The weighting function between all pairs of levels is of type M.1.

For recognition or classification problems using Supervised Learning, the NN is trained to reproduce given outputs from known input examples as accurately as possible. That is, the learning problem is defined by estimating the $\{w_{ij}\}$ such that a given output error, or "cost", function E is minimized. The most commonly used cost

function is the sum of the squares of the errors of the output nodes:

$$\min_{w_{ij}} \left\{ E = \frac{1}{2} \sum_{l} \sum_{c} (y_{lc} - t_{lc})^2 \right\} \tag{1.3}$$

where w_{ij} refers to all connection weights over all levels of the network, c indexes the input-output exemplar pairs, l indexes the output nodes, y_{lc} is the actual output, and t_{lc} is the desired output.

Though described in a variety of ways, learning techniques use either deterministic gradient descent (in particular, backpropagation [Rumelhart, Hinton and Williams, 1986b; Rumelhart, Hinton and Williams, 1986a; Plaut and Hinton, 1987] or stochastic relaxation methods [*e.g.* Metropolis, Rosenbluth, Rosenbluth, Teller and Teller, 1953; Kirkpatrick, Jr. and Vecchi, 1983; Aarts and Korst, 1989] to find a set of weights (a state of the network) that satisfies Equation 1.3. Solution or convergence times can be improved with various acceleration techniques [*e.g.* Fahlman, 1988].

We are not so much concerned with these search techniques but, rather, the definition of the state space being searched. The state space dimensionality is very high since it is defined by the total number of independent connection weights, which here includes every connection. Consequently, there are likely to be many local minima available to the solution technique that satisfy Equation 1.3 for the training examples, but do not characterize the problem in general. The problems associated with search time (number of iterations for convergence, *etc.*) are also significant. Further, under the classical NN perspective, no further constraints are added to such estimation problems as that would challenge the idea that relations and rules can be "discovered" automatically through the minimization or search for global minima of Equation 1.3. Here, we propose to retain the notion that states can indeed be discovered by the learning procedure, but to constrain the search procedure by modeling the network to explicitly include our knowledge of the important features of the data, or desired invariant properties.

This modeling can take a variety of forms. The network is frequently divided into a number of *modules*. These modules can be designed to perform component sub-tasks of the overall classification problem. Within a module, the weights may be constrained in some way: some weights may be set by hand; others may be constrained by a functional relationship; the module may be trained to perform a mapping independent of the classification training data. First, however, this view must be put in perspective. To this end, we will first relate the parameter estimation problem in NNs to similar problems in Signal Processing and Principal Components Analysis.

1.1.2 A Matrix View of Neural Networks

We first note that, for a given set of connections, Equations 1.1 and 1.2 can be written in matrix form as:

$$X_n = f\{\mathbf{W}Y_m\} \tag{1.4}$$

where f corresponds to the logistic nonlinear transducer (Equation 1.2), $\mathbf{W}$ to the connection matrix of size $n \times m$ (for m "source" and n "destination" nodes), and X and Y to the source and destination vectors corresponding to the pair-wise NN node layers. This form points to the essential idea that the flow of information from one level to the next corresponds to a transformation, defined by the connection matrix, which maps m-dimensional signals into n-dimensional ones, where, usually, $n < m$. The nonlinear transducer is used to map input values monotonically to the range $(0, 1)$. The net effect is that the NN procedure endeavours to discover a mapping which satisfies an overt constraint like least squares (Equation 1.3) or others.

Consequently, the best form of $\mathbf{W}$ is that of an orthogonal mapping of rank equal to the true dimensionality of the input - as measured, for example, by the rank of the input signal correlation matrix. Hidden units (units in neither the input nor output layers) correspond to components of the eigenstructure in the mappings, albeit constrained by the optimization goal. Except for the nonlinear transducer function, the use of hidden units in NNs is a way of determining the eigenstructure, or principal components, of the data. There must be at least as many hidden units (spread throughout the hidden layers) as there are non-zero eigenvalues in the input data correlation matrix, since it is necessary to span a vector space of the same dimensionality as the signal samples. The hidden units, by definition, extract the important features in the signals which, in more traditional signal processing, are extracted through the eigenvectors of the signal autocorrelation or autocovariance matrices [Ade, 1983; Ahmed and Rao, 1975].

Since many applications of NNs lie in the area of classification, it is interesting to note that, when the final output layer of such a cascaded NN is a set of nodes corresponding to classes or categories, we can interpret the system as a form of Discriminant Function Analysis. The Discriminant Analysis model is based upon the determination of decision hyperplanes which lie between sample class means and which maximize the between-class and minimize the within-class variance from their projections onto such planes. Discriminant functions are linear equations of the form:

$$z = \sum_{i=1}^{m} a_i y_i + a_0 \tag{1.5}$$

and class membership is determined by the value of the function. For example, for a two-class classification problem, class membership is determined by the sign of z. For

the n-class problem Equation 1.5 generalizes to the matrix form:

$$Z = \mathbf{D}Y \tag{1.6}$$

where the n-class vector Z determines the weights associated with the data, Y, from each of the classes. $\mathbf{D}$ corresponds to the set of discriminant function coefficients which define each class. The crucial difference between NNs and classical Discriminant Function Analysis is that the decision boundries of NNs are non-linear, due to the transducer f.

Similar formulations for classification have occurred in the Pattern Recognition literature where, for example, the Least Squares Minimum Distance Classifier [Ahmed and Rao, 1975] attempts to find a mapping, in matrix form, which transforms samples in feature space into points in "decision space" (whose dimensionality corresponds to the number of classes) and satisfies the constraint that samples from the same class should be mapped as close as possible to each other, and as far away as possible from other class samples, in decision space.

However, NNs differ from these well-known past techniques, even in their traditional form, by binding the representation and processing characteristics together. Although all these techniques have similar aims and structures, the NN formulation integrates the processing characteristics with the decision processes in one network, which is represented, in general, by a set of cascaded transformations. The use of nonlinear transducers and layers of differing sizes has the disadvantage that analytic solutions are difficult, particularly since the dimensionality of the representation is of high order, but permits the network to distinguish between classes which are not linearly separable.

Our aim is to preserve this binding of *process* with *representation* (feature space), but to extend the NN philosophy to include more explicit constraints on the network geometry and connection weights. The resulting systems behave similarly to traditional NNs, but have two main advantages. First, it is possible to construct networks that are constrained to respond to features of the input data that are known *a priori* to characterize the task or to have desired invariances, rather than hoping that the training data will cause the optimization technique to find a set of weights with these properties. Secondly, this allows the dimensionality of the system to be reduced, which can reduce the chance of finding a local minimum that characterizes the training data but not the general task. We call this extension Model-Based Neural Networks (MBNN).

The use of MBNNs allows a network to be constructed in which the supervisor's knowledge of the task to be performed is used to specify, partially or completely, the roles of some hidden units, or of whole hidden layers or modules, in advance. Thus the supervisor's knowledge of which features of the training data are significant for the task is incorporated into the network geometry and connection weighting functions, serving

as a constraint on the state space searched.

In this thesis, the aim is to demonstrate the utility of the MBNN approach for the specific problem of invariant pattern recognition. To situate this work, a review of existing general techniques for invariant pattern recognition is presented in Chapter 2, and of previous neural network-based approaches in Chapter 3. Later chapters present a variety of different forms of MBNN, and demonstrate their utility on a range of invariant pattern recognition problems, both real and synthetic.

Chapter 5 introduces a new invariant feature of two-dimensional contours, the Invariance Signature. It is shown that a MBNN can be constructed which calculates this (multi-dimensional) feature. This MBNN can be equipped with a classification module which uses the Invariance Signature as the basis for classification. Chapter 7 demonstrates that such Invariance Signature-based MBNNs can be employed successfully for shift-, rotation- and scale-invariant optical character recognition.

Chapter 2

General Methods for Invariant Pattern Recognition

The ability to perceive the permanent features of the visual environment is something which humans take for granted. Indeed, it is hard to imagine a world in which we could not. We do not even notice that we recognize objects and patterns independently of changes in lighting conditions, shifts of the object or observer, or changes in orientation and scale. [Gibson, 1966][1] stated this rather well:

> It can be shown that the easily measured variables of stimulus energy, the intensity of light, sound, odor, and touch, for example, vary from place to place and from time to time as the individual goes about his business in the environment. The stimulation of receptors and the presumed sensations, therefore, are variable and changing in the extreme, unless they are experimentally controlled in a laboratory. The unanswered question of sense perception is how an observer, animal or human, can obtain constant perceptions in everyday life on the basis of these continually changing sensations. For the fact is that animals and men do perceive and respond to the permanent features of the environment as well as to the changes in it.
>
> Besides the changes in stimuli from place to place and from time to time, it can also be shown that certain higher-order variables of stimulus energy – ratios and proportions, for example – do not change. They remain invariant with movements of the observer and with changes in the intensity of stimulation. And it will be shown that these invariants of the energy flux at the receptors of an organism correspond to the permanent properties of the environment.
>
> The active observer gets invariant perceptions despite varying sensations. He perceives a constant object by vision despite changing sensations of light; he perceives a constant object by feel despite changing sensations of pressure; he perceives the same source of sound despite changing sensations of loudness in his ears. The hypothesis is that constant perception depends on the ability of the individual to

[1] quote taken from [Wechsler, 1990, p. 96].

> detect the invariants, and that he ordinarily pays no attention whatever to the flux of changing sensations.

In this thesis we are concerned with only a small subset of the problems outlined above: the invariant perception of two-dimensional patterns under shift, rotation and scaling in the plane. This corresponds to the ability of humans to recognize patterns such as typed or handwritten characters independently of their size, orientation or position, which they do unthinkingly when reading a document such as an architectural drawing.

An image in the Cartesian (x, y) domain may be defined as a function $f : \mathbb{R}^2 \to \mathbb{R}$, $f(x, y) = z$, where z is the intensity of the image at coordinates (x, y). The invariant pattern recognition problem is to recognize images as being in some sense "the same" even though they have undergone a variety of allowed transformations. In two-dimensional pattern recognition, one typically desires invariance with respect to a combination of some or all of the shift, rotation and scale transformations.

A transformation may be denoted by $T_{\mathbf{a}}$, where $\mathbf{a}$ is a vector of parameters specifying the transformation. The deformed image is denoted by $f_{T_{\mathbf{a}}} = f\left(T_{\mathbf{a}}x, T_{\mathbf{a}}y\right)$. If the image is not invariant under the action of the transformation, then in general $f_{T_{\mathbf{a}}} \neq f_{T_{\mathbf{a}'}}$ if $\mathbf{a} \neq \mathbf{a}'$. The problem is thus to find a means of identifying f and $f_{T_{\mathbf{a}}}$ as instances of the same image.

Many different approaches to this problem have been used in computer vision, and more specifically in the application of neural networks to invariant pattern recognition. Some involve finding representations of the image in spaces other than the natural (x, y) space such that the representation in the new space is invariant under the desired transformations. Others search for specific features in the image which define "parts", and then seek invariant relationships between these parts. Still others fit analytic curves to parts of the image, and use algebraic or differential invariants of the parameters of these curves. In this chapter an attempt will be made to review these approaches, and to identify their strengths and weaknesses.

2.1 Integral Transforms

The most general form of an integral transform of an image $f(x, y)$ is

$$\mathscr{O}\left[f(x,y)\right] = F(u,v) = \iint_{\mathbb{R}^2} f(x,y)K(u,v;x,y)dxdy, \tag{2.1}$$

where $K(u, v; x, y)$ is the *kernel* of the transform, and $F(u, v)$ is the representation of the image in the transform domain (u, v). This operation is also frequently described as *filtering* the signal $f(x, y)$. $K(u, v; x, y)$ is the point spread function of the filter, and $F(u, v)$ is the filtered signal.

Integral transforms are useful for invariant pattern recognition if it is possible to choose kernels so that the transformed images are invariant (or have invariant components) under some specified transformations.

It is frequently the case that integral transforms used in invariant pattern recognition have two components:

- one that is invariant under the action of some transformation
- one that encodes the transformation with respect to some origin.

Invariant matching can be achieved using the first component, while the second can be used to recover the transformation. If the transformation can be inverted to recover the original image, then the representation is unique. If the invariant component is unique up to the specified transformations, then the representation can be called "invariant in the strong sense". Invariant, but not unique, representations can be said to be "invariant in the weak sense" [Ferraro and Caelli, 1994].

These two components are frequently the amplitude and phase spectra of the complex transform of the image.[2] The transform may be written as

$$\mathscr{O}\left[f\left(x,y\right)\right] = A\left(u,v\right)e^{-i\phi(u,v)}. \tag{2.2}$$

Considering a one-parameter transformation T_a, the conditions for strong invariance can be met if

$$\mathscr{O}\left[f_{T_a}(x,y)\right] = A\left(u,v\right)e^{-i[\phi(u,v)+ua]}, \tag{2.3}$$

since the amplitude spectrum A is invariant under the transformation T_a (*i.e.* it does not depend on a), and the parameter a of the transformation can be recovered from the phase spectrum. The only further requirement is that $A(u,v)$ is unique for each image.

There are limitations on the construction of strongly-invariant integral transforms, which will be discussed in the following matter. Whether strongly-invariant representations are necessary, or even desirable, is a moot point, especially since in many pattern recognition tasks one wishes to classify a pattern as a member of a class, rather than to recognize uniquely each individual pattern.

[2] Throughout this thesis, the symbol i will be used to denote $\sqrt[+]{-1}$, except when it appears as a subscript, where it indicates an indexing variable.

2.1.1 The Fourier Transform

Consider a one-dimensional function $f(x)$. Its Fourier transform is

$$F(u) = \int_{-\infty}^{\infty} f(x)e^{-iux}dx. \tag{2.4}$$

We denote the operation of taking the Fourier transform by the operator $\mathscr{F}$, so that we can write $F(u) = \mathscr{F}f(x)$. Now consider a version of the function f which is transformed by the one-parameter translation T_a specified by a, $f_{T_a}(x) = f(x-a)$. Taking the Fourier transform of $f_{T_a}(x)$, we obtain

$$\mathscr{F}f_{T_a}(x) = \int_{-\infty}^{\infty} f(x-a)e^{-iux}dx. \tag{2.5}$$

Changing variables to $x' = x - a$,

$$\begin{aligned}\mathscr{F}f_{T_a}(x) &= \int_{-\infty}^{\infty} f(x')e^{-iu(x'+a)}dx' \\ &= e^{-iua}\int_{-\infty}^{\infty} f(x')e^{-iux'}dx' \\ &= e^{-iua}\mathscr{F}f(x).\end{aligned} \tag{2.6}$$

Thus we see that a shift in the spatial domain produces only a phase-shift in the complex Fourier frequency domain. Taking the magnitudes of the transforms, we have

$$\left|\mathscr{F}f_{T_a}(x)\right| = \left|\mathscr{F}f(x)\right|. \tag{2.7}$$

The Fourier transform thus has two important properties:

- The magnitude of the Fourier transform is invariant under shifts in the spatial domain.[3]
- The shift a can be recovered, as it is encoded in the phase.

The Fourier transform is thus strongly-invariant with respect to shifts. These properties extend to multidimensional Fourier transforms [Rosenfeld and Kak, 1982], and form the basis of many invariant pattern recognition techniques [for example, Altmann and Reitböck, 1984; Lin and Brandt, 1993; Caelli and Liu, 1988; Ferraro and Caelli, 1988; Ferraro and Caelli, 1994; Rubinstein, Segman and Zeevi, 1991; Segman, Rubinstein and Zeevi, 1992].

[3]The amplitude of the power spectrum is thus invariant. Since the Fraunhofer (far-field) diffraction pattern of an aperture is just its two-dimensional Fourier transform, and the intensity of light corresponds to its power, a marvelous demonstration of this invariance can be carried out with a laser, lenses and an optical bench.

2.1.2 The Fourier-Mellin Transform

The Mellin[4] transform of a function $f(x)$ defined over the positive reals is the complex function $M(s)$, where

$$M(s) = \int_0^\infty f(x)x^{s-1}dx. \tag{2.8}$$

The Mellin transform of a function is closely related to its Fourier transform, and this is important for its application to invariant pattern recognition. If we change variables in Equation 2.8 to

$$x' = \frac{-\ln(x)}{i}, \tag{2.9}$$

we obtain

$$M(s) = -i\int_{-\infty}^\infty f(e^{-ix'})e^{-isx'}dx'. \tag{2.10}$$

Equation 2.10 shows that the Mellin transform of a function over the positive reals is, up to a multiplicative constant, just the Fourier transform of the function after the coordinate has been logarithmically-distorted. This means that the magnitude of the Mellin transform will be invariant under shifts in the logarithmically-distorted domain.

To see the utility of this property for invariant pattern recognition, consider a representation of an image in polar coordinates: $r \in [0, \infty)$ and $\theta \in [0, 2\pi)$. The domain of r makes it a suitable candidate for the application of the Mellin transform. Thus, we change variables to $\rho = \frac{-\ln(r)}{i}$. We know that the magnitude of the Mellin transform will be invariant under transformations $T_\mathbf{a}$ of the image such that $T_\mathbf{a}\rho = \rho + \alpha$, where α is some constant.

Such a transformation is a scaling of the image by some positive constant λ. We see that

$$T_\mathbf{a}r = \lambda r, \tag{2.11}$$

so that

$$\begin{aligned} T_\mathbf{a}\rho &= \frac{-\ln(\lambda r)}{i} \\ &= \frac{-(\ln(r) + \ln(\lambda))}{i} \\ &= \rho + \alpha \end{aligned} \tag{2.12}$$

[4] Hjalmar Mellin, 1854-1933.

where $\alpha = \frac{-\ln(\lambda)}{i}$. Thus the magnitude of the Mellin transform of the radial coordinate of the image is invariant under scaling of the image.

We observe that a rotation of image by an angle ϕ corresponds to a shift in the angular coordinate θ:

$$\theta' = \theta + \phi. \tag{2.13}$$

Thus the magnitude of the Fourier transform of the angular coordinate will be invariant under rotations of the image. The image representation given jointly by the Mellin transform of the radial coordinate and the Fourier transform of the angular coordinate is thus invariant under rotation and scaling of the image. This is the Fourier-Mellin transform, which has been widely employed for rotation- and scale-invariant pattern recognition.

Biological Instances of Log-Polar Mapping

The notion of changing a scaling to a shift by a logarithmic transformation of the coordinates occurs in many areas. In fact, the log-polar coordinate system described above seems to have a biological analogue. Altmann and Reitböck (1984, p. 46) say that

> The principle of logarithmic mapping seems to play a role in biological sensory systems too. In the simian visual system, as well as in the visual system of the cat, the mapping of the central $20° - 30°$ of retinal space onto area 17 of the visual cortex approximates a polar coordinate transformation together with a logarithmic distortion of the r-axis.

Since biological systems can shift objects to the centre of the field of view by movement of the eyes, it seems logical that a mapping that facilitates scale and rotation invariant recognition would be most useful. This is not to say that biological systems also compute Fourier transforms of these representations. Other methods can take advantage of representations in which transformations are reduced to shifts, as will be discussed in §2.3.

Other Invariances Using the Fourier and Mellin Transforms

We have seen above that a scale- and rotation-invariant representation of an image can be obtained by taking the Mellin transform of its radial coordinate and the Fourier transform of its angular coordinate. By combining these transforms in other ways, it is possible to obtain representations of images which are invariant under other transformations. As an example, we will consider Altmann and Reitböck (1984), who obtain a scale- and shift-invariant representation by taking the absolute value of the Mellin transform of the normalized Fourier amplitude spectrum of an image.

We have seen that the Fourier amplitude spectrum is shift-invariant. It is easy to show that a coordinate scaling in the image domain ($x' = \alpha x$) produces a coordinate scaling[5] and a change in magnitude in the Fourier domain. The magnitude change can be removed by normalizing the amplitude spectrum by its magnitude at $u = 0$. This normalized Fourier amplitude spectrum is thus shift-invariant, and changes only in scale when the image is scaled. The Mellin transform of this will thus be invariant under both shifts and scalings of the image.

It is important to note that, since the Fourier phase spectrum is discarded and the amplitude is normalized, this representation is not unique, and therefore cannot be inverted. It thus does not meet the criteria for strong invariance. Nevertheless, Altmann and Reitböck (1984) show that it can still be a useful representation. Whatever the properties of the exact, continuous transform derived in theory, these will not be preserved in a discrete digital implementation. Exact invariance is not preserved due to sampling and border effects, so *strong invariance* can *never* be achieved in a discrete system. This is a limitation common to all digital invariant pattern matching methods.

Altmann and Reitböck (1984) derive criteria for the preservation of "essential information" in the discrete Fourier transform of an image. For an $N \times N$ image, the total object size should be $n_t \lesssim N/4$, and the smallest permissible detail size is $n_d \gtrsim 4$, measured in pixels. Of course, even once suitable images have been obtained, and the invariant representations computed, the problem of matching these representations to those of the template patterns remains.

2.1.3 Canonical Coordinates

We have seen above that the notion of transforming a pattern into a space in which the action of a transformation $T_{\mathbf{a}}$ reduces to a shift along a coordinate axis is useful for invariant pattern recognition. Indeed, it turns out that this notion can be generalized into a scheme for computing the kernel $K(u, v; x, y)$ of an integral transform which will map an image into a space in which specified transformations are reduced to shifts, and which satisfies the strong invariance condition. This is possible if we consider the transformations of the image to be Lie transformation groups.[6] This is not overly restrictive, as most transformations of interest for (especially two-dimensional) invariant pattern recognition (*e.g.* shift, rotation and scaling) are Lie transformation groups. It will be seen that the coordinates in which the transformations reduce to shifts are the *canonical coordinates* of the generators of the transformation groups.

This formalism was introduced by Ferraro and Caelli (1988), and independently by Rubinstein et al. (1991). The results are extended, developed and applied in Segman et al. (1992) and Ferraro and Caelli (1994). The development presented here is an

[5] Dilations in the image domain become contractions in the frequency domain, and *vice versa*.

[6] The theory of Lie transformation groups is recapitulated in Chapter 5.

amalgam of these papers. We have seen that the conditions for strong invariance with respect to a transformation T_a, as given by Equation 2.3, can be satisfied if we find a change of coordinates $(x, y) \to (\xi, \eta)$ such that T_a becomes $\widetilde{T}_a$, a *shift*, in the new coordinate system (ξ, η).

Canonical Coordinates for a One-Parameter Transformation

Consider a one-parameter Lie Transformation Group T_a specified by

$$\begin{aligned} x' &= \alpha\,(x, y, a) \\ y' &= \beta\,(x, y, a)\,. \end{aligned} \tag{2.14}$$

Let $\mathscr{L}_T$ be the generator of T in the original (x, y) coordinates:

$$\mathscr{L}_T = \mu\,(x, y)\,\frac{\partial}{\partial x} + \nu\,(x, y)\,\frac{\partial}{\partial y}. \tag{2.15}$$

The action of T_a in the new coordinates is specified by

$$\begin{aligned} \xi' &= \xi\,(\alpha(x, y, a), \beta(x, y, a)) = \bar{\alpha}\,(\xi, \eta, a) \\ \eta' &= \eta\,(\alpha(x, y, a), \beta(x, y, a)) = \bar{\beta}\,(\xi, \eta, a)\,, \end{aligned} \tag{2.16}$$

and the new generator is

$$\widetilde{\mathscr{L}}_T = \left(\mu\frac{\partial \xi}{\partial x} + \nu\frac{\partial \xi}{\partial y}\right)\frac{\partial}{\partial \xi} + \left(\mu\frac{\partial \eta}{\partial x} + \nu\frac{\partial \eta}{\partial y}\right)\frac{\partial}{\partial \eta}. \tag{2.17}$$

In order to meet the strong invariance condition, we require that $\widetilde{G}$ be a shift in the new coordinates:

$$\begin{aligned} \xi' &= \xi + a \\ \eta' &= \eta. \end{aligned} \tag{2.18}$$

The generator of the transformation corresponding to this shift along the ξ-axis is

$$\widetilde{\mathscr{L}}_T = \frac{\partial}{\partial \xi}. \tag{2.19}$$

Equation 2.17 thus leads to the equations

$$\begin{aligned} \mu(x, y)\frac{\partial \xi}{\partial x} + \nu(x, y)\frac{\partial \xi}{\partial y} &= 1 \\ \mu(x, y)\frac{\partial \eta}{\partial x} + \nu(x, y)\frac{\partial \eta}{\partial y} &= 0. \end{aligned} \tag{2.20}$$

The new coordinates (ξ, η) specified by Equations 2.20 are called the *canonical coordinates* corresponding to the transformation T_a. It is well-known that this system of partial differential equations has an infinite number of solutions [see Rubinstein et al., 1991].

To apply this result to the derivation of a kernel for an integral transform, we recall that the Fourier transform satisfies the strong invariance condition with respect to along-axis shifts in its coordinate system. We now show how to combine the solutions to Equation 2.20 with the normal two-dimensional Fourier transform to find a new invariance kernel. Let $\widetilde{f}(\xi, \eta)$ be a function defined on $\mathbb{R}^2$. Its two-dimensional Fourier transform is

$$\widetilde{F}(u, v) = \iint_{\mathbb{R}^2} e^{i(u\xi + v\eta)} \widetilde{f}(\xi, \eta) d\xi d\eta. \tag{2.21}$$

The transformation T_a changes $\widetilde{f}$ according to $\widetilde{f}_{T_a}(\xi, \eta) = \widetilde{f}(\xi + a, \eta)$. We know from Equation 2.6, therefore, that

$$\widetilde{F}_{T_a}(u, v) = e^{-iua} \widetilde{F}(u, v). \tag{2.22}$$

Changing back to the original coordinates in Equation 2.21, we obtain

$$\begin{aligned} \widetilde{F}(u, v)[f] &= \iint_{\mathbb{R}^2} e^{i(u\xi(x,y) + v\eta(x,y))} J(x, y) f(x, y) dx dy \\ &= e^{-iua} \widetilde{F}(u, v)[f_{T_a}], \end{aligned} \tag{2.23}$$

where

$$J(x, y) = \det \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \xi}{\partial y} \\ \frac{\partial \eta}{\partial x} & \frac{\partial \eta}{\partial y} \end{bmatrix} \tag{2.24}$$

is the Jacobian of the change of coordinates. Thus we have found the kernel

$$K(u, v; x, y) = e^{i(u\xi(x,y) + v\eta(x,y))} J(x, y) \tag{2.25}$$

which satisfies the strong invariance conditions of Equation 2.3. Since Equations 2.20 have an infinite number of solutions, there are infinitely many invariance kernels for the one-parameter group G. This allows a particular kernel to be chosen so that it is convenient from a computational point of view.

Canonical Coordinates for Two Transformations

The method above can be extended to find canonical coordinates for two-parameter Lie transformation groups.[7] Unlike the one-parameter case, in which infinitely many kernels giving integral transforms satisfying the strong invariance conditions could be found, in the two parameter case the method leads to unique coordinate changes, and to the discovery of important restrictions on the existence of strongly-invariant transforms.

In order to satisfy the strong invariance condition simultaneously for two Lie transformation groups T_a and S_b, we require a representation

$$\mathscr{O}\left\{S_b\left[T_a f(x,y)\right]\right\} = A(u,v)\, e^{i\phi(u,v)+ua+vb}. \tag{2.26}$$

Again, the amplitude is invariant under the action of the transformations. The parameters the transformations are encoded *orthogonally* in shifts along the two axes in the transformation domain, in the phase.

Ferraro and Caelli (1994) derive the conditions under which such a representation exists, and specify the analytical form of the related integral transform kernel. If $\mathscr{L}_a$ and $\mathscr{L}_b$ are the generators of T_a and S_b respectively, then (ξ,η) are canonical coordinates for the transformations if

$$\begin{aligned} \mathscr{L}_a\xi &= 1, \qquad \mathscr{L}_a\eta = 0, \\ \mathscr{L}_b\xi &= 0, \qquad \mathscr{L}_b\eta = 1. \end{aligned} \tag{2.27}$$

The existence of such canonical coordinates is a sufficient condition for the existence of a strongly-invariant representation, as a simple consequence of the properties of the Fourier transform. It can also be shown that this condition is necessary [Ferraro and Caelli, 1988]. Canonical coordinates only exist if $\mathscr{L}_a$ and $\mathscr{L}_b$ are linearly independent and commute, that is, the Lie bracket vanishes:

$$[\mathscr{L}_a, \mathscr{L}_b] = \mathscr{L}_a\mathscr{L}_b - \mathscr{L}_b\mathscr{L}_a = 0. \tag{2.28}$$

Moreover, the change of coordinates is one-to-one. Equation 2.28 allows one to determine whether or not two transformations admit a strongly-invariant representation via a straightforward calculation. For example, consider rotation and dilation. The generator of the rotation transformation is

$$\mathscr{L}_R = -y\frac{\partial}{\partial x} + x\frac{\partial}{\partial y} \tag{2.29}$$

[7] This analysis is extended to k-parameter groups in Segman et al. (1992).

and that for dilation is

$$\mathscr{L}_D = x\frac{\partial}{\partial x} + y\frac{\partial}{\partial y}. \tag{2.30}$$

By simple calculation,

$$\begin{aligned}\mathscr{L}_R\mathscr{L}_D &= -y\frac{\partial}{\partial x}\left(x\frac{\partial}{\partial x} + y\frac{\partial}{\partial y}\right) + x\frac{\partial}{\partial y}\left(x\frac{\partial}{\partial x} + y\frac{\partial}{\partial y}\right) \\ &= -y\frac{\partial}{\partial x} + x\frac{\partial}{\partial y}.\end{aligned} \tag{2.31}$$

Similarly,

$$\begin{aligned}\mathscr{L}_D\mathscr{L}_R &= x\frac{\partial}{\partial x}\left(-y\frac{\partial}{\partial x} + x\frac{\partial}{\partial y}\right) + y\frac{\partial}{\partial y}\left(-y\frac{\partial}{\partial x} + x\frac{\partial}{\partial y}\right) \\ &= x\frac{\partial}{\partial y} - y\frac{\partial}{\partial x}.\end{aligned} \tag{2.32}$$

Trivially, the commutator is

$$[\mathscr{L}_R, \mathscr{L}_D] = 0. \tag{2.33}$$

Since $\mathscr{L}_R$ and $\mathscr{L}_D$ are orthogonal, and we have just shown that they commute, there exists a representation which is simultaneously strongly-invariant under both the corresponding transformations.

Conversely, consider $\mathscr{L}_X = \frac{\partial}{\partial x}$, corresponding to a translation along the x-axis. It is easy to show that the commutator with the rotation transformation is

$$[\mathscr{L}_X, \mathscr{L}_R] = \frac{\partial}{\partial y} \neq 0. \tag{2.34}$$

Similarly, $[\mathscr{L}_X, \mathscr{L}_D] \neq 0$, and the same is clearly true for translations along the y-axis. Thus Equation 2.28 allows us to show that there exists no representation which is strongly-invariant under both rotation and shift, or under dilation and shift. This refutes some claims in the literature [*e.g.* Casasent and Psaltis, 1976], and explains why the representations of Altmann and Reitböck (1984) discussed above could not be *strongly*-invariant.

The second benefit of the canonical coordinates approach is that it provides an equation for the kernel of the integral transform giving the strongly-invariant representation. This is of exactly the same form as that found above in Equation 2.25, with the condition that the commutation and independence conditions above must now be applied to the two transformation groups with respect to which (ξ, η) are canonical coordinates. An excellent treatment of this, and a generalization to k-parameter groups,

is given by Segman et al. (1992).

We have seen that the canonical coordinates approach provides a formalism which allows one to determine for which transformations a strongly-invariant integral transform representation exists, and moreover, when it exists, it shows how to calculate its kernel. These are important and powerful results.

2.2 Moments

Although calculated by integrating a kernel over an image, moments are not integral transforms in the sense used above. An integral transform transforms an image into a space of the same dimensionality as the image space (although the transform coordinates are often complex). A moment of a given order $(p+q)$, however, is a single number. If one thinks of the "moment transform" of an image as the set of all its moments, then one can consider the image to have been mapped into a discrete "moment space", indexed by p and q. In this sense moments can be considered as integral transforms.

2.2.1 Geometrical Moments

The geometrical moment of order $(p+q)$ of an continuous image function $f(x,y)$ is defined by:

$$m_{pq} = \iint_{\mathbb{R}^2} x^p y^q f(x,y) dx dy \qquad p,q = 0,1,2,\dots \,. \tag{2.35}$$

For invariance purposes, it is more convenient to use the central moments, which are, by definition, translation invariant:

$$\mu_{pq} = \iint_{\mathbb{R}^2} (x-\bar{x})^p (y-\bar{y})^q f(x,y) dx dy \qquad p,q = 0,1,2,\dots \,, \tag{2.36}$$

where the centroid of the image can be found using $(\bar{x},\bar{y}) = (m_{10}/m_{00}, m_{01}/m_{00})$. Altmann and Reitböck (1984) use μ_{20} and μ_{02} to estimate the factor by which an image has been scaled. This is then used as the starting point for a cross-correlation-based search for the image template in a reduced search space.

If the image is scaled according to $x' = \alpha x$, $y' = \alpha y$, a set of normalized central moments can be defined by the equations

$$\begin{aligned} \mu'_{pq} &= \frac{\mu_{pq}}{\alpha^{p+q+2}} \\ \eta_{pq} &= \frac{\mu'_{pq}}{{\mu'_{00}}^{\gamma}}, \qquad \gamma = \frac{p+q}{2} + 1. \end{aligned} \tag{2.37}$$

These normalized central moments are scale invariant, and the $\{\eta_{pq}\}$ can be used to define a set of moment features which are invariant under shifts, rotations and scalings of the image. Such a set [Khotanzad and Lu, 1990; Jain, 1989][8] is:

$$\phi_1 = \eta_{20} + \eta_{02} \tag{2.38a}$$

$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \tag{2.38b}$$

$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \tag{2.38c}$$

$$\phi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \tag{2.38d}$$

$$\begin{aligned}\phi_5 = {} & (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right] \\ & + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right]\end{aligned} \tag{2.38e}$$

$$\phi_6 = (\eta_{20} - \eta_{02})\left[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{03} + \eta_{21}) \tag{2.38f}$$

$$\begin{aligned}\phi_7 = {} & (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right] \\ & - (\eta_{30} - 3\eta_{21})(\eta_{12} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right].\end{aligned} \tag{2.38g}$$

Functions ϕ_1 through ϕ_6 are also invariant under reflection, while ϕ_7 changes sign.

Wechsler (1990) notes that these moments are not fault-tolerant, and in general produce disappointing results when used in pattern recognition experiments. The major sources of deviation from theoretical invariance are discretization, and quantization noise. They are also defeated by occlusion (as are many of the techniques discussed in this chapter). It is possible to define moments in polar coordinates too:

$$\psi_{kpq} = \int_0^\infty \int_{-\pi}^{\pi} r^k g(r,\theta) \cos^p\theta \sin^q\theta \, d\theta \, dr. \tag{2.39}$$

This allows a somewhat simpler expression of Equations 2.38 (and a very much simpler proof of rotation invariance), but the drawbacks remain the same.

2.2.2 Zernike Moments

Geometrical moments, as defined by Equation 2.35, have the form of a projection of the image $f(x,y)$ onto the monomial $x^p y^q$. The basis set $\{x^p y^q\}$ lacks some desirable features, such as orthogonality. Consequently, the resultant geometrical moment features are sub-optimal in terms of information redundancy [Khotanzad and Lu, 1990].

The Zernike polynomials $\{V_{nm}(x,y)\}$ are a set of complex polynomials which form

[8] Both sources cite Hu (1962) as the originator of these invariant moment combinations, but I have not obtained that paper.

a complete orthogonal set over the interior of the unit circle:

$$V_{nm}(x,y) = V_{nm}(r,\theta) = R_{nm}(r)e^{im\theta} \qquad \begin{cases} n \in \mathbb{J},\ n \geq 0 \\ m \in \mathbb{J},\ n - |m| \text{ even},\ |m| \leq n \end{cases} \tag{2.40}$$

where

$$R_{nm}(r) = \sum_{s=0}^{n-\frac{|m|}{2}} \frac{(-1)^s[(n-s)!]r^{n-2s}}{s!\left(\frac{n+|m|}{2}-s\right)!\left(\frac{n-|m|}{2}-s\right)!}. \tag{2.41}$$

These polynomials, unlike the monomials $x^p y^q$, are orthogonal, satisfying

$$\iint\limits_{x^2+y^2\leq 1} V_{nm}^*(x,y)V_{pq}(x,y)dxdy = \frac{\pi}{n+1}\delta_{np}\delta_{mq}, \tag{2.42}$$

where δ_{ab} is the usual Kronecker delta: $\delta_{ab} = 1$ if $a = b$, and 0 otherwise.

The Zernike moments $\{A_{nm}\}$ of an image $f(x,y)$ are the projections of the image onto these basis functions. Expressed in polar coordinates,

$$A_{nm} = \frac{n+1}{\pi} \iint\limits_{r\leq 1} f(r,\theta)V_{nm}^*(r,\theta)drd\theta. \tag{2.43}$$

The orthogonality property means that the image can be reconstructed from these moments (albeit an infinite number of them). To compute the Zernike moments for a digital image, the usual summation approximation of the integral is used. The centre of the image is taken as the origin, and the pixel coordinates are scaled to the unit circle. Pixels falling outside the unit circle are discarded [Khotanzad and Lu, 1990].

The Zernike moments provide rotation-invariant features. Consider the Zernike moment A_{nm} of an image, and A'_{nm}, the Zernike moment of the image after rotation through an angle ϕ. Since the radial and angular components of the Zernike polynomials are separable, it is trivial to note that

$$A'_{nm} = A_{nm}e^{-im\phi}. \tag{2.44}$$

This is reminiscent of the property of the Fourier transform discussed in §2.1.2. It is clear then that the magnitudes of the Zernike moments are a set of rotationally-invariant image features. This invariance is of course not exact in a digital setting due to the usual problems of digitization and quantization noise, but Khotanzad and Lu (1990) show σ/μ values[9] of less than 7% for a range of Zernike moments for variously rotated images of real characters.

[9] σ is sample standard deviation, μ is sample mean.

Zernike moments are only rotation-invariant. To obtain shift and scale invariance, the images must first be normalized with respect to these actions. This is done by using the geometric moments of the raw image. In short, the normalized image is

$$f'(x,y) = f\left(\bar{x} + \frac{x}{a}, \bar{y} + \frac{y}{a}\right) \tag{2.45}$$

where $(\bar{x}, \bar{y}) = (m_{10}/m_{00}, m_{01}/m_{00})$ is the image centroid, and $a = \sqrt{\beta/m_{00}}$, where β is a pre-determined image "size" (in pixels, for a binary image). This normalization renders $|A_{00}|$ identical for all images, and $|A_{11}| \equiv 0$.

Khotanzad and Lu (1990) compared a number of classifiers (multilayer perceptron (MLP), Bayes, nearest-neighbour and minimum mean distance) using features based on geometrical moments and Zernike moments for classifying arbitrarily transformed uppercase Roman letters. They found that the neural network classifier outperformed all the others, especially in noisy cases. In agreement with Wechsler (1990), they found that the geometric moment features became useless in the presence of noise, whereas the Zernike moment features proved robust, although the higher order moments were more sensitive. Perantonis and Lisboa (1992) compared a MLP trained with Zernike moment features with a third-order network (see §3.4.3) for classifying hand-written numerals, and found that the third-order network performed better. Higher-order neural networks are, however, extremely expensive in space and computation time.

2.3 Matched Filtering and Convolution Techniques

The integral transform techniques discussed in §2.1 address the problem of finding a representation of an image which is invariant under certain transformations of the image. The amplitude spectra obtained by these techniques, the invariant part of the representation, are themselves images. The "Invariant" part of the phrase "Invariant Pattern Recognition" has been addressed, but not yet the actual "Recognition". Matched filtering provides one answer to the recognition problem, and convolution, or cross-correlation, techniques employ it to provide invariance as well.

2.3.1 Matched Filtering

One of the oldest techniques for pattern recognition is matched filtering, descriptions of which can be found in many standard reference books on computer vision [Rosenfeld and Kak, 1982; Ballard and Brown, 1982]. Matched filtering allows the computation of R_{fg}, a measure of the similarity, between an image $f(x,y)$ and a template $g(x,y)$:

$$R_{fg} = \iint f(x,y)g(x,y)dxdy, \tag{2.46}$$

where the integration is over the entire domains of x and y.

If the image and template are normalized so that they have the same total intensity *i.e.* $\iint f^2(x,y)dxdy = \iint g^2(x,y)dxdy$, then R_{fg} is proportional to the interesting part of the mean-squared distance between the images d^2_{fg}, since

$$\begin{aligned} d^2_{fg} &= \iint (f(x,y) - g(x,y))^2 dxdy \\ &= \iint \left(f^2(x,y) - 2f(x,y)g(x,y) + g^2(x,y)\right) dxdy \\ &= 2\iint f^2(x,y)dxdy - 2R_{fg}. \end{aligned} \tag{2.47}$$

The first term is constant due to the normalization, so it is clear that R_{fg} measures the least-squared "similarity" between the image and the template: the larger R_{fg} is, the closer the images are to identical. In a pattern recognition problem, for an image and each of a set of possible templates, R_{fg} is computed and the image is assigned to the class of the template for which R_{fg} is greatest. No classification may be made if R_{fg} is less than some threshold. It is known that this is the optimal technique for detecting templates in the presence of additive white noise [Rosenfeld and Kak, 1982].

The matched filter output is extremely sensitive to shifts or distortions of the image and it is rarely used in this simple form. Nevertheless, it is a candidate for the in-place matching of the amplitude spectra obtained using the integral transforms described in §2.1.

2.3.2 Cross-correlation

A more common approach is to compute the cross-correlation $C_{fg}(u,v)$ between the image and a template, frequently with a template smaller than the image:

$$C_{fg}(u,v) = \iint f(x,y)g(x+u, y+v)dxdy. \tag{2.48}$$

$C_{fg}(u,v)$ is again an "image", in which the intensity at coordinates (u,v) is just R_{fg} calculated for the template shifted to those coordinates before the comparison with $f(x,y)$.

The presence of the template $g(x,y)$ centred at (u,v) in the image $f(x,y)$ can be detected if $C_{fg}(u,v)$ is greater than some threshold at that point. It is possible to detect the presence of multiple instances of a template in the image using this technique. Some version of cross-correlation plays a role in many invariant pattern recognition techniques [Altmann and Reitböck, 1984; Caelli and Liu, 1988; Pintsov, 1989; Zetzsche and Caelli, 1989; Lenz, 1990; Ohlsson, 1992; Lin and Brandt, 1993].

The use of cross-correlation with raw images and templates can be seen as a form of

shift-invariant pattern recognition, since the template can be detected no matter where it is in the image. It is perhaps not a true invariant pattern recognition technique, since the image is effectively exhaustively-searched for the template. Computing $C_{fg}(u, v)$ is a very computationally-expensive process, especially if the template is large.

Pintsov (1989) uses the idea of "the cross-correlation function with respect to a group of transformations". He views Equation 2.48 as cross-correlation with respect to the group of translations of the plane:

$$\begin{aligned} x' &= x + u, \\ y' &= y + v. \end{aligned} \tag{2.49}$$

The cross-correlation process calculates R_{fg} between the image and all possible versions of the template transformed by the group of translations. This notion can be extended to other groups. In general, if a group G has n parameters, then locating instances of a transformed template in an image by means of cross-correlation is the problem of finding the local maxima of the cross-correlation function in an n-dimensional space. Pintsov (1989) shows how this approach leads directly to the Hough transform which is discussed in detail in §2.5.1, and the closely-related Radon transform. It is noted that the computational cost may be reduced if the transformation has known symmetries.

Extending the cross-correlation function to higher dimensions in this way as a means of achieving invariance was discussed earlier by Caelli and Liu (1988, p. 207), but, as they noted:

> ... if in the cross-correlation process, we explicitly include α and θ as variables, (by generating all possible templates) then the correlation process is invariant with respect to rotation (θ) and size change (α) as well as translation (x, y) in a four dimensional space. However, the large amount of computation involved in calculating the quadruple integration ... prohibits the process from being practical, even with the help of fast Fourier transform techniques.

Again, this is really just exhaustive search: all possible transformations of the template are computed and compared with the image. Pintsov (1989) shows how known symmetries of transformations can be used to reduce the space which must be searched. Altmann and Reitböck (1984) give a method that reduces this cost by only searching for the template in regions of the image (in their case actually a Fourier transform of an image) which are likely to contain it. This is possible because they are using a Fourier-Mellin technique in which scale changes are mapped to shifts in the transform domain, and the normalized central second moments (see §2.2) are used to estimate the scale factor.

2.3.3 Filters Based On Multiple Templates

We have seen that computing a cross-correlation is equivalent to doing a matched filtering of the image with all possible transformed versions of a template. The computational cost increases dramatically as the number of parameters of the allowed transformation group increases, rendering this approach largely impractical. This cost can be greatly reduced if it is necessary to compute the matched filter output of the image with only a small set of transformed patterns. This is possible, for instance, if the number of possible transformations of the image is known *a priori*.

One approach is to use a filter H_j which is a linear combination of the n allowed transform states f_{ij} of the jth pattern:

$$H_j = a_{1j}f_{1j} + a_{2j}f_{2j} + \cdots + a_{nj}f_{nj}, \tag{2.50}$$

where the $\{a_{ij}\}$ are determined by some suitable constraints. In order to achieve simultaneous invariance to translation, rotation and scaling, a very large number of templates is still required, and finding the $\{a_{ij}\}$ is again computationally-expensive. This approach can be effective, however, if the template design procedure is based on some optimization criteria.

Caelli and Liu (1988) address this problem by defining the "invariance surface"[10] of an image. The coordinates of an pattern $f_j(x, y)$ may be rotated and scaled, giving new coordinates (ξ, η) according to

$$\begin{bmatrix} \xi \\ \eta \end{bmatrix} = \alpha \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}. \tag{2.51}$$

The invariance surface $I_j(\alpha, \theta)$ for pattern j is the maximum of the shift cross-correlation function between the untransformed pattern and the rotated and scaled versions,

$$I_j(\alpha, \theta) = \max_{u,v} \left\{ \iint_{\mathbb{R}^2} f_j(\xi, \eta) f_j(x + u, y + v) dx dy \right\}. \tag{2.52}$$

The degree of invariance of a particular pattern with respect to the transformations is given by $I_j(\alpha, \theta)$: the degree of invariance is measured by the difference between the cross-correlation function and its maximum value, when computed between the untransformed template and itself. The degree of invariance of a pattern depends on the pattern structure: the letter "O" is much more rotationally-invariant than the letter "T", and its $I_j(1, \theta)$ curve is correspondingly smoother, and deviates less from $\max_\theta\{I_j(1, \theta)\}$. This indicates that the number of templates required to construct a

[10] This is a different use of this term than that used later in this thesis.

satisfactory filter H_j depends of the invariance surface of pattern j. This number can be determined by choosing a recognition threshold criterion C, and choosing to generate a new template at each transformation state (α, θ) where $I_j(\alpha, \theta)$ falls below C. Clearly this will automatically take account of any symmetries of the patterns, and of the transformations.

This technique allows a computationally feasible application of the matched filtering method to invariant pattern recognition. The number of templates used to construct the filter for each pattern depends on the invariance properties of the pattern with respect to the transformations for which invariance is desired. Moreover, this can be controlled by manipulating the threshold C, trading matching accuracy for computational expense.

Zetzsche and Caelli (1989) employ a related method. They use a 4-dimensional pattern representation based on banks of filters. The representation is encapsulated in a set of filtered images, each containing the original Cartesian coordinates of the image, and indexed in terms of the orientation- and scale-specific filters used to produce them. This is based on transformations to a space in which rotations and scalings become shifts, as discussed extensively above, and again claims biological inspiration. Rather than filters based on transformed target images, the filters are in this case generic, based on two-dimensional Gabor functions, which are discussed in Chapter 4. This is thus not a direct matched filtering technique, but cross-correlation is used for recognition of the filter outputs.

2.4 Parts and Relationships

In all the techniques discussed so far in this chapter, an attempt has been made to find an invariant representation of the image as a whole, and to match images to images (or transformed images) on a pixel-by-pixel basis. A different approach to the invariant pattern recognition problem is to view an image as a group of component *parts*, and *relationships* between those parts. As a trivial example, if an image consists of two straight lines, then the angle between those lines is invariant under shifts, rotations and scalings of the image. The lines are the parts, and the angle between them is their relationship. Matching is done between these abstracted properties of the image, rather than between pixels. In more realistic examples, there is the sub-problem of (possibly invariantly) recognizing the parts in the image, and extracting their relationships.

There is a large body of literature concerning the application of the parts and relationships approach to three-dimensional object recognition. Here we will consider only a small sample of two-dimensional invariant pattern recognition applications. It should be noted that the distinction between a "holistic" pattern recognition approach and the use of parts and relationships is not as clear-cut as it might seem. Holistic approaches depend implicitly on the constituent parts of the image, and their relationships.

2.4.1 Sub-Graph Matching

Li (1992) addresses the problem of matching between an image and a model under arbitrary translations, rotations and scale changes in noisy conditions. Only very simple objects (or patterns) are considered, the parts being points or line segments. An image can consist of only sub-parts of a model object, and there can be extra lines and points in the image. Li (1992) calls his object representation an *attributed relational structure* (ARS). An ARS consists of a set of nodes, their properties (unary relations), and binary and ternary relations between them. The ARS is an invariant image representation, because the node properties and relations are chosen to be invariant under arbitrary 2D geometric transformation. Possible choices for these invariant attributes include curvature properties, relative angles and distance and area ratios.

It is apparent that, once parts and relationships have been extracted from an image, the parts and relationships constitute a graph, with labeled nodes and edges. This is true of all parts and relationships approaches to object or pattern recognition, whether or not the attributes chosen are invariant. The greatest limitation of these techniques is thus that recognition requires a solution of the sub-graph isomorphism problem, which is known to be NP-complete [Ballard and Brown, 1982]. The challenge is thus to find an algorithm that can obtain an acceptable solution in reasonable time. Explicit search may be satisfactory for sufficiently small graphs. Another approach is relaxation labeling, which is employed by Li (1992).

2.4.2 Deformable Templates

Less general, but related, are the deformable template techniques. Typically they are used when the aim is to match a distorted version of a known model, for which the possible distortions are limited and non-arbitrary. These conditions allow costs to be assigned to various deformations, and constraints to be placed on the matching problem. One advantage of these techniques is that they not only allow matching of distorted templates, but allow the distortion to be recovered, and its cost to be estimated.

Chow and Li (1993) apply deformable template matching to the automatic detection of features in front-view images of human faces. Domain knowledge, such that the eyes are a horizontal pair of dark regions on a light surface, is used to provide initial plausible estimates of feature locations, for features such as the eyebrows and mouth. A generalized Hough transform is then used to locate the irises, modeled as circles. These estimated iris positions are in turn used as a starting point for the application of a deformable template, consisting of two parabolas, to fitting the eye boundaries. The optimization method used is based on the simplex method, which allows the reduced search space to be taken into account [Press, Teukolsky, Vetterling and Flannery, 1992]. A similar method is used to locate the mouth. Experiments with this system showed

that it was able to locate facial features satisfactorily in over 80% of test images.

Deformable template techniques share several features with the model-based methods developed in this thesis. The system designer's *a priori* knowledge of the domain of possible patterns is used to constrain the search for features. This domain knowledge is directly encoded in the system, rather than being abstracted from training images.

2.5 Contour-Based Methods

An important class of two-dimensional invariant pattern recognition techniques, especially in the context of this thesis, are those designed for the recognition of contours. These contours may be extracted through some form of edge detection, or may be a natural pattern representation, such as in character recognition. Broadly speaking, approaches to contour recognition may be divided into two classes: those which represent the contour by a parameterized algebraic expression fitted to the image, and those which treat a contour as a group of pixels. Contour recognition is of particular interest because contours corresponding to object boundaries are frequently used in three-dimensional object recognition, and also because there are many applications for the application of machine pattern recognition to domains in which the patterns naturally consist of line drawings (*e.g.* character recognition, circuit diagrams, engineering drawings, *etc.*). Moreover, there is evidence that the human visual system applies a contour-based approach to pattern recognition even when no contour actually exists in the image: it interpolates an implied contour [Caelli, Preston and Howell, 1978].

2.5.1 The Hough Transform

Perhaps the simplest "contour-based" technique is the Hough transform, which is described in standard computer vision references [*e.g.* Ballard and Brown, 1982]. Variants of the Hough transform occur frequently in the invariant pattern recognition literature [Pintsov, 1989; Chow and Li, 1993; Li and Roeder, 1994]. We have seen in §2.3.2 that the Hough transform and its generalizations can be interpreted as cross-correlation techniques, where the template is a parametric curve rather than an actual image. The most basic Hough transform is that for detecting straight lines.

Consider an image in which the intensity $I(x, y)$ at a point (x, y) is a measure of that point's importance (typically the "strength" assigned to it by an edge detection algorithm). All straight lines passing through (x, y) obey the equation $y = mx+c$. Each individual line is represented by a point (m, c) in a two-dimensional parameter space. The $\{(m, c)\}$ corresponding to all possible lines passing through (x, y) form a line in the parameter space. This parameter space is unbounded, so in practical applications,

the parameterization

$$x\cos\theta + y\sin\theta = r \tag{2.53}$$

is usually used. This produces a sinusoid in (r, θ) space for each point (x, y). The parameter space can be divided into a number of bins, or accumulators, each corresponding to a range of values of r and θ. For each point (x, y), $I(x, y)$ is added to the bins corresponding to all the (r, θ) satisfying Equation 2.53. As with cross-correlation, a threshold can then be used to chose which bins have accumulated sufficient evidence for the presence of a line.

The Hough transform in this form is usually thought of as a line detection algorithm, rather than an invariant pattern recognition technique. It can, however, be considered to be a method for recognizing a straight line segment invariant under shifts and rotations.[11] The use of the Hough transform for detecting more complex parameterized contours is more readily recognizable as invariant pattern recognition. The circles $(x-a)^2+(y-b)^2 = r^2$ can be detected in exactly the same way, this time with a three-dimensional parameter space (a, b, r). This form of the Hough transform is used by Chow and Li (1993), and, slightly modified, by Li and Roeder (1994).

The Hough transform is thus a generalized matched filtering technique for parameterized curves. It can be used to discover the set of parameters that best explain the image data, given that the parametric form of the curve producing the data is *a priori* known. Its disadvantage is that it is practically restricted to a reasonably small set of template curves, since both the computation time and the size of the array of bins increase exponentially with the number of parameters. Moreover, it is heavily dependent on the quality of the edge detection which produces $I(x, y)$.

2.5.2 Algebraic and Differential Invariants

Another approach to invariant contour matching involves calculating invariants of the contour, and matching these. This offers the chance to avoid the computation time and space expenses of methods such as cross-correlation or the Hough transform, in which the contour is effectively matched against all possible transformed versions of the template.

Such techniques are often employed when seeking three-dimensional projective invariants of contours, a more difficult problem than invariance under the subset of affine transformations in the plane discussed so far. The researchers involved in this work are frequently interested in issues such as camera calibration, stereo matching or photometrics, rather than with invariant object (or pattern) recognition *per se* [Forsyth,

[11] The choice of threshold influences the scale invariance of this technique.

Mundy and Zisserman, 1992; Weiss, 1993b; Faugeras, 1993; Barrett, Gheen and Payton, 1993; Gros, 1993].

Functions of Contour Coefficients

Algebraic invariants are well suited for use with algebraic contours: contours which can be expressed by an implicit polynomial $f(x, y) = 0$. As an example we will consider the family of contours known as conic sections. The general equation for a conic section is

$$f(x, y) = a_{11}x^2 + a_{22}y^2 + 2a_{12}xy + 2a_{13}x + 2a_{23}y + a_{33} = 0. \tag{2.54}$$

If we define the matrices

$$\mathbf{X} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix}. \tag{2.55}$$

Equation 2.54 can then be written succinctly in matrix form as

$$\mathbf{X}^{\mathrm{T}}\mathbf{A}\mathbf{X} = 0. \tag{2.56}$$

The shape of a particular conic is determined by the coefficients $\{a_{ij}\}$. Since $\mathbf{A}$ is a real symmetric matrix, the coordinates can be transformed using a similarity transform so that $\mathbf{A}$ is diagonal. This implies that properties of the matrix $\mathbf{A}$ which are invariant under similarity transforms will be invariant descriptors of the shape of the conic under translation and rotation. One such feature is the determinant $\mathbf{D}$, another is the trace $\mathbf{T}$. Two invariant descriptions of the conic $f(x, y)$ are thus

$$\mathbf{D} = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{vmatrix} \qquad \mathbf{T} = a_{11} + a_{22} + a_{33}. \tag{2.57}$$

Further invariant properties of $\mathbf{A}$ can be found.[12] These functions of the polynomial coefficients can therefore be used for matching a transformed conic to its untransformed template. This approach can be extended to higher dimensions (*e.g.* quartic surfaces), and to other families of curves. The coefficients must be obtained by fitting a polynomial to the image data – in itself a far from trivial problem, and potentially computationally-expensive. Clearly a high resolution image of the curve will be required if the coefficients are to be estimated sufficiently well to be useful for recognition, although Forsyth et al. (1992, p. 43) say "For applications in model-based vision, it is far more important that a representation be projectively invariant than that it be a good approximation." If

[12] For instance, any symmetric function of the eigenvalues.

the coefficients can be determined sufficiently well, however, the matching process is very cheap.

Cross-ratios

The techniques discussed in this section are often more applicable to the projective geometry appropriate in three-dimensional object recognition than in two-dimensional shift, rotation and scale invariance. Projective transformations preserve neither lengths, nor ratios of lengths. The ratio of two ratios of lengths, however, *is* invariant [Wechsler, 1990]. Such a ratio is known as a *cross-ratio*. Since projective transformations are many-to-one, matching cross-ratios are a necessary, but not sufficient, condition for feature-matching. Cross-ratios are used in many recognition problems involving projective geometry [*e.g.* Forsyth et al., 1992; Barrett et al., 1993; Vanderkooy, 1996].

An example of a possible cross-ratio that might be used comes from Vanderkooy (1996). Consider the detection of four coplanar lines, A, B, C and D. Under a perspective projection, these lines will intersect at a single point: the vanishing point.[13] The cross-ratio of the sines of the angles between these lines is:

$$[A, B; C, D] = \frac{\sin(\angle AC)}{\sin(\angle AD)} \times \frac{\sin(\angle BD)}{\sin(\angle BC)}. \tag{2.58}$$

If the lines are in fact coplanar, this corresponds to an invariant structure. If the lines are not coplanar, the cross-ratio will vary from view to view.

Note that this requires that the lines be labeled, so that the appropriate angles can be identified. This requires some prior point-matching. If this is not done, then the technique faces a combinatorial explosion. The number of cross-ratios, C, that can be calculated for N lines is

$$C = \frac{N!}{(N-4)!4!}. \tag{2.59}$$

Differential Invariants

Differential Invariants arise most naturally when the coordinates of points on a curve, $\mathbf{x}$, are expressed explicitly as a function of some local parameter t, $\mathbf{x} = \mathbf{x}(t)$, rather than by an implicit function such as that in Equation 2.54. The natural shape descriptors in such a representation are the derivatives $\frac{d^n x_i}{dt^n}$. These descriptors are *local*, since they depend on the derivatives at a particular value of t, unlike the global descriptors derived from the coefficients of the implicit function in the case of algebraic invariants. A differential invariant is a function of the $\frac{d^n x_i}{dt^n}$ which does not change under a transformation of the coordinates $\mathbf{x}$ and the parameter t (or which changes in a limited way, which will be

[13] Lines can be extended to the vanishing point; they need not intersect in the actual image.

explained in the following matter) [Weiss, 1993a]. Various differential invariants have been widely applied in computer vision: curvature, torsion and Gaussian curvature, for instance, are all invariant under Euclidean transformations [Forsyth et al., 1992].

It is possible to characterize a curve using the differential equation to which the curve is one of the possible solutions. The advantage of this approach is that some constants are eliminated. Curves are represented parametrically, $\mathbf{x} = \mathbf{x}(t)$, where $\mathbf{x}$ is a point on the curve, expressed in homogeneous coordinates.

As a simple example, consider the second order linear differential equation

$$\mathbf{x}'' = 0. \tag{2.60}$$

The solutions to this equation constitute the family of all straight lines. Equation 2.60 is thus an invariant representation of straight lines – it is invariant with respect to the gradient and y-intercept of the line (*i.e.* shift, rotation and scale).

Generalizing, any curve expressed in n-dimensional homogeneous coordinates satisfies the linear differential equation [Weiss, 1993b]:

$$\mathbf{x}^{(n)} + \binom{n}{1} p_1 \mathbf{x}^{(n-1)} + \binom{n}{2} p_2 \mathbf{x}^{(n-2)} + \cdots + p_n \mathbf{x} = 0. \tag{2.61}$$

It is clear that multiplying $\mathbf{x}$ by a constant matrix $\mathbf{T}$ will have no effect on Equation 2.61, since $\mathbf{T}$ will factor out. The p_i are thus invariant under the linear transformation $\mathbf{T}$. The solutions are curves in $(n-1)$-dimensional space, determined up to projection. The p_i in Equation 2.61 are scalar functions of the parameter t. Projective invariants can be defined in terms of these p_i.

For plane curves in 3-space, the differential equation corresponding to Equation 2.61 is

$$\mathbf{x}''' + 3p_1\mathbf{x}'' + 3p_2\mathbf{x}' + p_3\mathbf{x} = 0. \tag{2.62}$$

In order deal with coordinate scaling, it is necessary to define *semi-invariants*, P_2 and P_3, which are not invariant under changes in parameterization:

$$\begin{aligned} P_2 &= p_2 - p_1^2 - p_1' \\ P_3 &= p_3 - 3p_1p_2 + 2p_1^3 - p_1''. \end{aligned} \tag{2.63}$$

P_2 and P_3 are invariant under projection and coordinate scaling. We now need to find quantities invariant under transformations of the curve parameter t. We may write this transformation as $t \to \tilde{t}(t)$. *Relative invariants of weight* w are defined to be quantities

which transform as:

$$\tilde{\theta}_w = \frac{1}{(\tilde{t}')^w}\theta_w. \tag{2.64}$$

Two such relative invariants can be found using the quantities defined in Equation 2.63:

$$\begin{aligned}\theta_3 &= P_3 - \frac{3}{2}P_2' \\ \theta_8 &= 6\theta_3\theta_3'' - 7(\theta_3')^2 - 27P_2\theta_3^2.\end{aligned} \tag{2.65}$$

Absolute invariants ($w = 0$) can be found by combining relative invariants. For plane curves, an absolute invariant is θ_3^8/θ_8^4. This approach can be extended to higher dimensions.

One advantage of differential invariants is that they are complete – a small set of invariants contains all the essential information about the curve. They are also local, in that differential properties at one point determine the entire curve. This means that differential invariants are invulnerable to occlusion.

Whilst mathematically elegant, this approach has one great disadvantage. Its application to digital images requires the computation of extremely high-order derivatives of the contour in the image (as high as the eighth). This process is well-known to be error-prone. Moreover, these derivatives are raised to high powers, magnifying the estimation error.

Weiss (1993b), noting that the method above is unreliable and hard to use in practice, defines some modified semi-invariants which reduce the number of derivatives required. He also introduces a filter-based method for improving the accuracy of the estimation of high derivatives. Nonetheless, the results presented are for noiseless synthetic curves of very much higher resolution than those typically available for pattern recognition.

Chapter 3

Neural Network Approaches to Generalization and Invariance

In Chapter 2 several general methods for performing invariant pattern recognition were described. In this chapter, we move from the general to the specific, and consider in particular the application of *artificial neural networks* to the problem of invariant pattern recognition. This will involve a clarification of the terms "generalization" and "invariance"; both are frequently used in the neural networks literature, often as if they described the same phenomenon: this is not the case.

From its earliest history, practitioners in the field of neural networks have been interested in the pattern recognition problem. Perhaps this is due to the simple fact that the input layer of an artificial neural network is such a seductive analogue of the the biological retinal array of photoreceptors. This correspondence, coupled with the amount of the cortex devoted to vision, has long made pattern recognition a favoured area of inquiry for neural networks researchers. Indeed it could be argued that pattern recognition has been the major application domain for artificial neural networks since their invention, and some form of invariance has often been sought. The longevity of this quest, and how little it has changed to this day, is exemplified by Pitts and McCulloch (1947, p. 127), who, referring to biological nets, say:

> Numerous nets, embodied in special nervous structures, serve to classify information according to useful common characters. In vision they detect the equivalence of apparitions related by similarity and congruence, like those of a single physical thing seen from various places. ... The equivalent apparitions in all cases share a common figure and define a group of transformations that take the equivalents into one another but preserve the figure invariant. ... We seek general methods for designing nervous nets which recognize figures in such a way as to produce the same output for every input belonging to the figure.

Pitts and McCulloch (1947) were especially concerned with designing neural networks

which were consistent with what was known about actual mammalian cortical structure. Nevertheless, their models for translation and dilation invariance have much in common with more modern work.

In his seminal paper Rosenblatt (1958, p. 386) starts with the question of "the capability of higher organisms for perceptual recognition, generalization, recall and thinking", but segues seamlessly within two paragraphs to the "stored pattern", as if the storage and recognition of patterns were such a natural way to consider these problems that no explanation was necessary. Moreover, Rosenblatt uses a "photo-perceptron" as his example, again emphasizing the fact that visual pattern recognition has long been considered a prime application for neural networks. Rosenblatt's (1958, p. 404) confidence is exemplified by:

> The question may well be raised at this point of where the perceptron's capabilities actually stop, We have seen that the system described is sufficient for pattern recognition, associative learning, and such cognitive sets as are necessary for selective attention and selective recall. The system appears to be potentially capable of temporal pattern recognition, as well as spatial recognition, involving any sensory modality or combination of modalities.

When Minsky and Papert (1969) showed this confidence to be misplaced, it was done in the context of pattern recognition. Their proofs of the limitations of perceptrons in general were couched in the language of geometry: could finite-order or diameter-limited perceptrons compute *connectedness*? Could they compute *parity* or *convexivity*? Indeed they made explicit their interest in invariance and pattern recognition, saying "we are adopting the mathematical viewpoint of Felix Klein: every interesting geometrical property is an invariant of some transformation group" [Minsky and Papert, 1969, p. 41].

Kohonen's (1972) quite general model for a linear associative memory was demonstrated with character recognition examples. Amari (1977) also used pattern recognition as the problem domain for his more complex self-organizing associator, which addresses the association of non-orthogonal patterns which are a difficulty for correlation-matrix-based associators.

Grossberg (1980), although addressing the general problem of the development of internal representations of the environment through experience, uses geometric pattern-matching as his example. This paper contains much of the theory, including the notion of feedback-induced resonance between input patterns and stored prototypes, which forms the basis of of his later *Adaptive Resonance Theory (ART)*, the subject of many papers [*e.g.* Carpenter and Grossberg, 1987; Carpenter and Grossberg, 1988].

Despite the activity mentioned above, the field of artificial neural network research had in reality been dealt a severe blow by Minsky and Papert's (1969) critique. It was the popularization of the backpropagation algorithm for training non-linear multilayer

perceptrons (MLPs) in the mid 1980s that sparked a new explosion of interest in the field. It is the non-linearity of the MLPs that allows them to escape the limitations of perceptrons demonstrated by Minsky and Papert (1969). Backpropagation, in reality a fairly simple application of the chain rule, was independently discovered by several researchers. It is its ingenious distributed implementation which makes it interesting, and applicable to neural networks. In one of the influential early papers on back-propagation, Rumelhart et al. (1986b) demonstrate its ability to solve "rather abstract mathematical problems" [Rumelhart et al., 1986b, p. 631] such as parity and encoding. When, however, they want a more realistic application they turn to a simple invariant pattern recognition problem: discriminating between a "T" and a "C" independent of translation and rotation.[1]

It should be clear that the history of artificial neural networks is inextricably bound up with their application to pattern recognition. There is a very large number of papers and books on this subject, and an attempt to survey them is beyond the scope of this thesis. Excellent treatments exist, such as Bishop (1995). The remainder of this chapter will thus address the application of neural networks to invariant pattern recognition. The somewhat nebulous notion of generalization will also be addressed, since it is often confounded with invariance. Invariance, in fact, is a particularly special case.

3.1 Quantifying Generalization

Since their earliest inception, many claims have been made about the ability of neural networks to generalize. In the context of pattern classification, this usually means that the network is able to assign correct class labels to patterns that were not presented to it during training. Class membership is usually to an extent subjective – whether, for example, a given pattern is an instance of a particular letter of the alphabet is an unsolved problem mathematically, and is in fact context-sensitive even for humans [McGraw, 1992]. Generalization is, in its simplest form, due to the fact that neural networks interpolate: small changes in the input usually produce small changes in the output [Amari, 1990]. This, however, is a far from satisfactory definition of generalization for mathematical analysis.

A network which performs true invariant pattern recognition is a quite different entity to one which simply interpolates in subjectively "sensible" ways. A true invariant pattern recognition network has an output that is *by definition* invariant under the application of certain transformations to the input layer. This property should ideally be independent of the composition of any training data applied to the network. It is possible to imagine a truly invariant network which does not generalize correctly according to certain subjective criteria.

[1] A simple problem, since the characters used consisted of only five pixels.

It is important to note that many of the results and claims concerning the ability of neural networks to generalize have been made for *unstructured input data*, frequently in the context of generalization in the presence of noise. This makes plain the essential difference between generalization and invariance: in the invariance case, new input patterns belonging to the same class as a training pattern are produced by a completely deterministic mathematical transformation; in many generalization studies they are produced by random perturbations of the training pattern. Several of these approaches to measuring and improving generalization will now be reviewed.

3.1.1 Hamming Distance

Amari (1990) considers the properties of correlation matrix associative memories trained to associate two sets of independently randomly generated n-dimensional bipolar vectors, $\{\mathbf{s}^\mu\}$ and $\{\mathbf{q}^\mu\}$. The normalized Hamming distance between an input vector $\mathbf{s}_i^\mu$ and a noisy version $\mathbf{s}_i^{\mu\prime}$ is

$$D_{\mathbf{s}^\mu}(\mathbf{s}_i^\mu, \mathbf{s}_i^{\mu\prime}) = \frac{1}{2n} \sum_{i=1}^{n} \left| \mathbf{s}_i^\mu - \mathbf{s}_i^{\mu\prime} \right|. \tag{3.1}$$

Let the output of the network for an input vector $\mathbf{x}$ be $T_W\mathbf{x}$. The generalization ability of the network may be measured by considering the distance $D_{\mathbf{q}^\mu}(\mathbf{q}_i^\mu, T_W\mathbf{s}_i^{\mu\prime})$. Amari's analysis shows that the expected value of this distance is

$$D_{\mathbf{q}^\mu} = \Phi\left(-\frac{1-2D_{\mathbf{s}^\mu}}{\sqrt{r}}\right), \tag{3.2}$$

where

$$\Phi(u) = \int_{-\infty}^{u} \frac{1}{\sqrt{2\pi}} e^{-v^2/2} dv, \tag{3.3}$$

and r is the ratio of the number of training examples to the input dimensionality. This is an explicit expression of the ability of the network to generalize. It shows that the degradation of performance is very graceful, with a catastrophic transition to $D_{\mathbf{q}^\mu} \approx 1$ at $D_{\mathbf{s}^\mu} = 0.5$. A similar analysis is given for recurrent autocorrelation associative memories in the same paper.

It must be remembered, however that this is a statistical result, obtained for random input and output vectors, and n sufficiently large for the central limit theorem to be applied. Moreover, the Hamming distance is a totally inappropriate metric for pattern similarity in the case of transformation invariance. A shifted pattern may have no pixels in common with its prototype. Its normalized Hamming distance from its prototype is thus 1, whereas its "shift invariance difference" is 0. Despite this, statistical measures

based on such metrics make up much of the literature on generalization, and are often critically compared to invariance techniques.

3.1.2 Comparison with a Teacher Network

Another way to quantify generalization is to consider the output of a network which is trained using the inputs and outputs of a "teacher" network with an identical architecture [Krogh and Hertz, 1991]. Since the architectures are identical, it is known that the "student" network is capable of emulating the teacher perfectly, given the correct weights. Krogh and Hertz (1991) analyze the dynamics of the training of a single linear neuron with $p = \alpha N$ of the 2^N possible input-output pairs generated by a teacher network. The inputs are generated randomly and independently. They define the generalization error F to be the squared difference between the student and teacher outputs, averaged over all 2^N possible binary inputs:

$$F = \frac{1}{N}\sum_{i=1}^{N}(u_i - w_i)^2, \tag{3.4}$$

where $\mathbf{u}$ and $\mathbf{w}$ are the teacher and student weight vectors respectively. F varies between 1 and 0 (perfect generalization) if the weight vectors are normalized to length $\sqrt{N}$.

Krogh and Hertz (1991) devote much space to the analysis of F as a function of time during training, $F(t)$. For the purposes of this thesis, the asymptotic results at infinite time are those of interest, since we are concerned with the generalization ability of trained networks. The result is:

$$\lim_{t\to\infty} F(t) = \begin{cases} 1-\alpha & \text{for } \alpha < 1, \\ 0 & \text{for } \alpha \geq 1. \end{cases} \tag{3.5}$$

It may at first seem surprising that perfect generalization occurs when only the potentially-tiny fraction $N/2^N$ of possible inputs are presented, but this is in fact entirely unremarkable, since only N points are required to determine an $(N-1)$-dimensional hyperplane, which is all that is being done in this simple linear system.

Krogh and Hertz (1991) also investigate the effect of adding a weight decay term (see §3.3.1) to the weight-update equation. Without a weight decay term, learning only takes place in the subspace of the weight space spanned by the training input patterns. If the initial weight vector is non-zero, components outside this subspace can cause errors. The use of weight decay causes that part of the initial weight vector orthogonal to the input pattern space to decay exponentially, thus eliminating this source of error.

Winther, Lautrup and Zhang (1995) also investigate generalization performance where the target is defined by an unknown teacher network. They investigate a learning scheme which makes use of some prior knowledge to improve generalization performance. They use a Bayesian approach, in which the prior information available is $\Pr(D|V)$, the probability that the training dataset D was generated by a teacher network with parameters V. They find that the learning rule they derive is optimal in the sense that it gives the best possible expected generalization error (defined as the expected value of the squared output error over all possible inputs) for this formulation.

3.1.3 The Effective Number of Parameters

Yet another definition of generalization performance is the expected test set error $\langle E_{\text{test}}(\lambda)\rangle_{\xi\xi'}$, where λ is a regularization parameter (see §3.3), ξ is the training set and ξ' is the test set [Moody, 1992]. Moody (1992) gives an analysis of the relationship between the expected test error and the expected training set error for nonlinear learning systems such as MLPs. His main result is

$$\langle E_{\text{test}}(\lambda)\rangle_{\xi\xi'} \approx \langle E_{\text{train}}(\lambda)\rangle_{\xi\xi'} + 2\sigma_{\text{eff}}^2 \frac{p_{\text{eff}}(\lambda)}{n}, \tag{3.6}$$

where n is the size of the training set, σ_{eff} is the effective noise variance in the response, and p_{eff} is the *effective number of model parameters*. The expectation values are taken over possible training and test sets, and the result is to second order. Equation 3.6 can be compared with the exact result for linear models using a sum of squares error function and no regularizer:[2]

$$\langle E_{\text{test}}\rangle_{\xi\xi'} = \langle E_{\text{train}}\rangle_{\xi\xi'} + 2\sigma^2 \frac{p}{n}, \tag{3.7}$$

It is of interest that p_{eff} is in general not equal to the true number of model parameters p. The effective number of parameters depends upon the amount of model bias (*e.g.* network architecture), model nonlinearity and on prior model preferences determined by the regularization parameter λ and the form of the regularizer.

Many techniques for improving generalization are based on an Occam's Razor heuristic: simple models are best. Weight decay (see §3.3.1) is motivated by the intuitive notion that it causes unnecessary weights (*i.e.* parameters) to be removed. Moody's (1992) analysis confirms this, and indeed the result is applicable to all regularized unbiased linear models: p_{eff} is a decreasing function of λ, with $p_{\text{eff}}(0) = p$ and $p_{\text{eff}}(\infty) = 0$. If the model is biased and nonlinear, then in general $p_{\text{eff}}(0) \neq p$.

[2] For a particular input vector sampling probability density. See Moody (1992, p. 851).

3.1.4 The Vapnik-Chervonenkis Dimension

Another measure frequently used in the analysis of generalization in artificial neural networks for classification problems is the Vapnik-Chervonenkis (VC) dimension. Here we will follow the development of [Holden and Anthony, 1992]. Quite a few definitions are required before we proceed.

Without loss of generality, we consider a neural network with one output node, with takes on the values 0 or 1. Any given such network computes a class $\mathscr{F}$ of functions $f_{\mathbf{w}} : \mathbb{R}^n \to \{0, 1\}$, the actual function computed depending on the weight vector $\mathbf{w}$. These functions effectively split the input space $\mathbb{R}^n$ in two.

Definition 3.1 *The* hypothesis $h_{\mathbf{w}}$ *associated with a function* $f_{\mathbf{w}}$ *is defined as the subset of* $\mathbb{R}^n$ *for which* $f_{\mathbf{w}} = 1$,

$$h_{\mathbf{w}} = \{\mathbf{x} \in \mathbb{R}^n | f_{\mathbf{w}} = 1\}. \tag{3.8}$$

The hypothesis space H *computed by the network is the set*

$$H = \left\{h_{\mathbf{w}} | \mathbf{w} \in \mathbb{R}^W\right\} \tag{3.9}$$

of all hypotheses where W *is the total number of weights in the network.*

The VC dimension can be regarded as a measure of the expressive capacity of the hypothesis space of a network.

Definition 3.2 *Given a set* $S \subseteq \mathbb{R}^n$ *and some function* $f_{\mathbf{w}} \in \mathscr{F}$, *the* dichotomy (S^+, S^-) *of* S *induced by* $f_{\mathbf{w}}$ *is defined to be the partition of* S *into the disjoint subsets* S^+ *and* S^- *where* $S^+ \cup S^- = S$ *and* $\mathbf{x} \in S^+$ *if* $f_{\mathbf{w}} = 1$, $\mathbf{x} \in S^-$ *if* $f_{\mathbf{w}} = 0$.

Definition 3.3 *Given a hypothesis space* H *and* $S \subseteq \mathbb{R}^n$, *we define* $\triangle_H(S)$ *as the set*

$$\triangle_H(S) = \{h \cap S | h \in H\}. \tag{3.10}$$

We say that S *is* shattered *by* H *if* $\triangle_H(S) = 2^S$ *where* 2^S *is the set of all subsets of* S.

We are now in a position to define the *growth function* and the VC dimension.

Definition 3.4 (Growth Function) *The growth function is defined on the set of positive integers as*

$$\triangle_H(i) = \max_{S \subseteq \mathbb{R}^n, |S| = i} |\triangle_H(S)|. \tag{3.11}$$

Definition 3.5 (Vapnik-Chervonenkis dimension) *The VC dimension* $\mathscr{V}(H)$ *of a hypothesis space* H *is the largest integer* i *such that* $\triangle_H(i) = 2^i$, *or infinity if no such* i *exists.*

The growth function thus gives the maximum number of dichotomies induced by $\mathscr{F}$ for any set of i points. The VC dimension is the size of the largest set of points which can be shattered by H. As an example, consider the set of simple linear discriminant functions:

$$\mathscr{F}_L = \left\{ \text{sign}\left[w_0 + w_1 x_1 + \cdots + w_n x_n\right] \;\middle|\; \mathbf{w} \in \mathbb{R}^{n+1} \right\}. \tag{3.12}$$

It is well-known that $\mathscr{V}(\mathscr{F}_L) = n+1$. Such exact results are difficult to obtain for more complex systems such as neural networks, though some bounds have been found. For multi-layered perceptrons with W weights and N hidden nodes, the class of functions $\mathscr{F}_{\text{MLP}}(W, N)$ has the bound [Baum and Haussler, 1989]:

$$\mathscr{V}(\mathscr{F}_{\text{MLP}}(W, N)) \leq 2W \log_2(eN). \tag{3.13}$$

Another result, commonly known as *Sauer's lemma*, provides an upper bound for the growth function, given the VC dimension of a class of functions $\mathscr{F}$ [Holden and Anthony, 1992].

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

$$\triangle_{\mathscr{F}}(k) \leq 1 + \sum_{i=1}^{d} \binom{k}{i}. \tag{3.14}$$

For finite $\mathscr{V}(\mathscr{F})$, another useful bound is

$$\triangle_{\mathscr{F}}(k) \leq k^{\mathscr{V}(\mathscr{F})} + 1. \tag{3.15}$$

The VC dimension can be applied to the analysis of the ability of neural networks to generalize. As discussed above, a neural network may be considered to compute a class $\mathscr{F}$ of functions. Training of the network can be regarded as a process which tries to find some $f_{\mathbf{w}} \in \mathscr{F}$ which is in some sense a "good approximation" to a target function f_T on the given set of training examples. If an input vector $\mathbf{x} \in \mathbb{R}^n$ is chosen randomly according to an arbitrary distribution P, we can define $\pi_{f_{\mathbf{w}}}$ to be the probability that the network agrees with the target function for this input vector:

$$\pi_{f_{\mathbf{w}}} = \Pr\left[f_{\mathbf{w}}(\mathbf{x}) = f_T(\mathbf{x})\right]. \tag{3.16}$$

Now consider a sequence of k training examples, $\{(\mathbf{x}_1, f_T(\mathbf{x}_1)), \ldots, (\mathbf{x}_k, f_T(\mathbf{x}_k))\}$, chosen at random according to P. Let $v_{f_{\mathbf{w}}}$ be the fraction of the inputs for which $f_{\mathbf{w}}$ agrees with f_T. Training consists of choosing a weight vector $\mathbf{w}$ according to the value of $v_{f_{\mathbf{w}}}$. Consequently, it is necessary to know if $v_{f_{\mathbf{w}}}$ converges to $\pi_{f_{\mathbf{w}}}$ uniformly for all $f_{\mathbf{w}} \in \mathscr{F}$

as k becomes large. If it does not, we may choose a $v_{f_\mathbf{w}}$ for which $\pi_{f_\mathbf{w}}$ is relatively low, and the probability of good generalization is thus also low.

An inequality due to Vapnik[3] gives a bound for the probability that there is a function $f_\mathbf{w} \in \mathscr{F}$ for which $\pi_{f_\mathbf{w}}$ and $v_{f_\mathbf{w}}$ are significantly different. For a given threshold value α,

$$\Pr\left[\sup_{f_\mathbf{w}\in\mathscr{F}} \frac{v_{f_\mathbf{w}} - \pi_{f_\mathbf{w}}}{\sqrt{1-\pi_{f_\mathbf{w}}}} > \alpha\right] \leq 4\triangle_{\mathscr{F}}(2k)e^{\frac{-\alpha^2 k}{4}}. \tag{3.17}$$

The significance of this result arises because for finite $\mathscr{V}(\mathscr{F})$, by Equation 3.15, the growth function $\triangle_{\mathscr{F}}(k)$ is bounded above by a polynomial function of k. Since the right-hand side of Equation 3.17 decays exponentially as a function of k, the generalization error can be made arbitrarily small by choosing a sufficiently large k. Moreover, Equation 3.17 provides a bound on the rate of convergence during training.

Summary

Although these are useful and powerful ways of characterizing the ability of neural networks to generalize, they all address the generalization problem from the perspective of a learning environment in which the input and output vectors are essentially drawn at random from some population, according to some probability distribution. This may be appropriate when attempting to estimate the ability of a network to generalize in the presence of noise with known statistics, or when trying to quantify the representational power of a given network architecture. Theoretical results such as these, however, would seem to have little bearing on the design of networks where the transformations of the input patterns are known *a priori*. This serves to emphasize the difference between generalization and invariance.

3.2 Improving Generalization

3.2.1 Large Training Sets

The most basic approach to improving the generalization performance of a neural network, however it might be defined, is to increase the size of the training set. Many of the results described in §3.1 provide estimates or bounds for the number of training examples required to achieve a specified generalization performance. In the most basic sense, it is desirable to have sufficient training examples to span the parameter space of the network (see §3.1.2).

The use of large training sets is such a ubiquitous feature of neural network applications that it would be possible to cite thousands of references. As examples of the sizes

[3] This is from Holden and Anthony (1992).

of training sets often employed, Plaut and Hinton (1987) use 250,000 training examples for a network with 54 input nodes and 2 output nodes. This is discussed in detail in §4.3. Fontaine and Shastri (1992) use 5450 training images for their digit recognition system. Khotanzad and Lu (1990) use 24 slightly perturbed images of each of the 26 letters of the alphabet for their character recognition system. This approach is almost universal in the neural networks literature.

One of the aims of the Model-Based Neural Network approach is to obviate the need for such large training sets. By designing networks with guaranteed invariance properties, the need to include transformed and perturbed images in the training set can be removed. This aim will form a part of much that follows in this thesis.

3.2.2 Cognitron and Neocognitron

One of the most durable families of neural networks for pattern recognition has been that of Cognitron [Fukushima, 1975] and its successor, Neocognitron [Fukushima, 1980]. Cognitron is a self-organizing multi-layered network that is trained using a Hebbian scheme [Hebb, 1949]. Much of the inspiration for Cognitron was biological, and Fukushima made some claims now known to be false (*e.g.* "it is known that the capability of a layered neural network is greatly enlarged if the number of the neural layers is increased." [Fukushima, 1975, p. 121].

The basic structure of the Cognitron is a series of cascade layers consisting of neurons with limited "receptive fields": a neuron in the layer immediately following the input layer would have connection only to input neurons in a limited spatial area of the input layer. As one progresses through the network, the receptive fields of neurons increase in size, the idea being that neurons in the final layer respond to features on the entire input layer. Cognitron was trained using a small number of simple synthetic digits and numerals, and some associative memory properties were demonstrated.

The relevance of these networks to this study dates from Neocognitron [Fukushima, 1980] Fukushima (1980, p. 193) claims "The network is self-organized by "learning without a teacher", and acquires an ability to recognize stimulus patterns based on the geometrical similarity (Gestalt) of their shapes without effected by their positions."(sic). With Neocognitron, the series of cascaded layers used by Cognitron was retained, but the connection patterns and weights of neurons were constrained to be identical within a given plane, each connected to a spatially-shifted region of the layer above. There could be several planes in a given layer. "Hence, all the cells in a single cell-plane have receptive fields of the same function, but at different positions" [Fukushima, 1980, p. 195]. This is conceptually similar to the MBNN systems introduced in Chapter 4. Neocognitron was trained with a single noise-free example of each of the digits {0,1,2,3,4}, and was shown to be able to classify some distorted and noisy examples correctly.

This work was extended to larger datasets in Fukushima, Miyake and Ito (1983) and Miyake and Fukushima (1984). Supervised learning was introduced into the model, along with other variations. The authors continued to make strong claims for the invariant pattern recognition capabilities of their systems, *e.g.* "each cell of the deepest layer of the network responds selectively to a specific stimulus pattern and is not affected by the distortion in shape or the shift in position of the pattern" [Fukushima et al., 1983, p. 826].

The claims for the invariant properties of the Neocognitron were made without rigorous mathematical analysis. Barnard and Casasent (1990) show that the performance of the Neocognitron *is not intrinsically shift-invariant*, and that any invariance comes as a trade-off with classification sensitivity. In fact, Neocognitron's apparent shift invariance is basically the result of summing the total energy of the input layer. Despite this, work has continued based on the assumption that Neocognitron *is* shift-invariant: Himes and Iñigo (1992) combine log-polar mapping with the Neocognitron in an attempt to produce a shift-, scale- and translation-invariant system, and [Li and Wu, 1993] introduce layers that produce rotated versions of the input pattern to add rotation invariance to Neocognitron.

3.2.3 Cascade-Correlation

The Cascade-Correlation algorithm, introduced by Fahlman and Lebiere (1990), is a combined architecture and supervised learning algorithm. Rather than just adjusting the connection weights in a pre-determined network structure, Cascade-Correlation starts with a minimal network (no hidden nodes), and adds new hidden nodes one after another. Once a node is trained, its input weights are never trained again. New nodes have all existing nodes as inputs, and are connected to all the outputs. This means that no backpropagation is necessary, since the only weights ever adjusted are between the output units and all the other nodes in the network: the simple Widrow-Hoff delta rule can be used [Widrow and Hoff, 1960].

When a new hidden unit is needed a pool of *candidate units* is created. The candidate units have input connections from all the input units and existing hidden units. The candidate units are trained to maximize the correlation between the candidate units' outputs and the residual output error of the network. When training is deemed to have converged, the candidate unit with the highest correlation with the output error is selected and installed as the new hidden unit. Its input-side weights are fixed, and the process is then repeated until the output error of the network is acceptable. The architecture is also extensible to recurrent networks [Fahlman, 1991].

Fahlman and Lebiere (1990, p. 12) state that "by training one unit at a time instead of training the whole network at once, we can speed up the learning process considerably, while still creating a reasonably small net that generalizes well". In terms of

generalization, Cascade-Correlation belongs to the "less is more" school. It builds a network that is near-minimal in terms of the number of nodes and connections necessary to characterize the training set. Many techniques for improving generalization (weight decay, pruning, *etc.*) are based on the notion that fewer network parameters leads to better performance.

The drawback is that in Cascade-Correlation *everything* about the network is determined by the training set. Such a technique must be vulnerable to overfitting noise in training sets unless the training sets are large. Cascade-Correlation does not address the problem of invariance at all. It is worth noting that the notion that smaller networks generalize better has been challenged. Lawrence, Giles and Tsoi (1996), for instance, report that oversize networks can result in lower training and generalization error for certain problems.

3.2.4 Massive Weight-Sharing

Lautrup, Hansen, Law, Mørch, Svarer and Strother (1994, p. 1) state the generalization problem very nicely:

> The aim of learning is to match a model to data in such a way that generalization ability ensues. Whether this is possible depends intricately on the training procedure and on the architecture of the learning machine. If the model is overly restrictive, it cannot "capture the rule", hence it fails to implement the the training set. On the other hand if we train a model with too high capacity for a given data set, it is unlikely that the model will generalize. The reason is that there will be many different ways to implement the training set in the model, *i.e.* to generalize from it.

Lautrup et al. (1994, p. 1) consider generalization for extremely ill-posed problems: problems with a vast number of highly-correlated inputs, but only a small number of available training patterns. Such problems are common in image and spectral analysis. Their solution uses a similar approach to that of Singular-Valued Decomposition [Press et al., 1992]: the problem is transposed from a high-dimensional input space to a low-dimensional "signal-space". In neural network terms, the effect is to induce *massive weight-sharing* by constraining the network weights to a low-dimensional subspace, rather than letting training explore the entire space of dimensionality equal to the number of weights. This is done by expanding the weights $\{\mathbf{w}\}$ so that they are expressed as linear combinations of the set of vectors $\{\mathbf{x}\}$ which span the signal space of training inputs.

$$\mathbf{w}_j = \sum_{\alpha=1}^{p} \gamma_j^{\alpha} \mathbf{x}_{\alpha} \tag{3.18}$$

Optimization is then performed on the coefficients $\{\gamma_j^\alpha\}$ rather than on the weights.

This "trick" is only possible if there are strong correlations between the components of the input vector. In fact this approach is directly related to Principal Components Analysis. Principal Components Analysis gives direct access to a set of orthogonal vectors which span the input space, and it is possible to formulate Massive Weight-Sharing to use these vectors as its basis. Principal Components Analysis also ranks these vectors in terms of their contributions to the input variance. It would be possible to reduce the dimensionality of the effective weight space by truncating the spanning set according to this ranking.

Lautrup et al. (1994) report that this technique was used to reduce the dimensionality of a problem from Positron Emission Tomography from 141,375 to 47. The generalization results reported are in fact not very good. This is attributed to the fact that most of the variance (the first 7 principal components) is in fact inter-subject variance.

3.2.5 Pruning Techniques

Another approach to reducing the dimensionality of a network's weight space in the hope that this will improve generalization is to "prune" weights which are deemed to be insignificant.[4] Rather than simply removing small weights by analogy with Weight Decay (see §3.3.1), pruning techniques take a more sophisticated approach. Optimal Brain Damage [Le Cun, Denker and Solla, 1989] removes the weights that have the least effect on the training error, based on a diagonal approximation of the Hessian of the network. The pruned network must be retrained, introducing additional computational cost. The diagonal assumption is also inaccurate [Asriel U. Levin and Moody, 1994]. Other pruning techniques exist, such as Principal Components Pruning [Asriel U. Levin and Moody, 1994], but all are training set-driven, and none address the invariance problem.

3.2.6 Using Prior Information

One of the aims of the Model-Based Neural Networks approach is to provide a mechanism by which the network designer can utilize his prior expert knowledge of desired network performance and the problem domain to constrain the construction of networks. Several existing approaches to improving network performance make use of prior information, in a variety of ways.

Omlin and Giles (1992) discuss the training of recurrent networks using "hints". By this they mean that some weights are set to large absolute values before training starts, rather than the usual initialization of all weights to small random values. Their

[4] To prune a weight means to remove the corresponding connection from the network.

application is the recognition of regular languages defined by transition grammars. Hints are inserted by setting weights corresponding to known allowed or forbidden transitions to large or small values respectively. Only partial knowledge of the target grammar is included as hints. They found that this prior knowledge could indeed improve convergence time during training, but that generalization performance with hints was unpredictable, and "did not suffer significantly by using hints" [Omlin and Giles, 1992, p. 6].

The ability of the backpropagation algorithm to make use of prior information about data from known distributions is investigated by Barnard and Botha (1993). They take a Bayesian perspective: the prior knowledge to which they refer is the probability that an unknown input pattern belongs to a particular class. They report that linear least mean-squares classifiers can be shown to place their decision boundaries sub-optimally on this basis: they tend to guess the most likely class too often. In this case the prior information is implicit in the training set, in the form of the frequency of class examples. They find that backpropagation on multi-layered networks tends to employ such prior information sub-optimally, although the difference from a theoretically-determined Bayes optimal classifier was slight. Experimental results indicate that network performance approaches the Bayes limit as the size of the training set increases. This is in contrast to the linear classifier, where the performance is a function of the values of the prior probabilities.

Prem, Mackinger and Dorffner (1993) introduce a method for inserting *a priori* symbolic knowledge into a neural network called *Concept Support*. Rather than relying upon setting particular weights, knowledge is inserted by pre-training the network on concepts or rules thought to be important for the eventual task to be performed by the network. It is hoped that this will address problems such as the fact that not all necessary knowledge might be present in the training data (for generalization), and that pre-programming might help the network to avoid spurious local minima during subsequent training. Prior knowledge inserted in this way can always be "over-written" during later training.

As an example of Concept Support, Prem et al. (1993) considered the problem of diagnosing coronary artery disease from thallium-201 scintigrams of the heart. Prior information could take the form of statements such as "there is a difference between the scans of men and women". A network was initially trained to classify scans as belonging to a man or a woman. The output units were then removed, and replaced by a new unit. The network was then trained to classify the input for the presence of disease on the left or the right side of the heart. The correct test set prediction rate rose from 60% to 73% as a result of this Concept Support. The experiments in Prem et al. (1993) all show improved performance as a result of this technique, but the treatment is entirely empirical.

3.3 Regularization Approaches to Invariance

The most common method used to attempt to improve generalization is to augment the usual least squares cost function E (Equation 1.3) with some auxiliary function of the weights, producing a new cost function C:

$$C = \frac{1}{2} \sum_{l} \sum_{c} (y_{lc} - t_{lc})^2 + \lambda g(w_{ij}). \tag{3.19}$$

The choice of this auxiliary function $g(w_{ij})$ often seems to be based more on intuition, or empirical knowledge of network performance, than any specific knowledge of the task that the network is supposed to perform. The training process then becomes a compromise between minimizing Equation 1.3 and minimizing the auxiliary function. The auxiliary function is often a "complexity measure" of the network. Consequently, there is an underlying heuristic in this approach: less "complex" networks generalize better.

3.3.1 Weight Decay

Probably the most common auxiliary function is the sum of the squares of all the weights in the network [Krogh and Hertz, 1992; Plaut and Hinton, 1987]:

$$C = \frac{1}{2} \sum_{l} \sum_{c} (y_{lc} - t_{lc})^2 + \lambda \frac{1}{2} \sum_{i} w_{ij}^2. \tag{3.20}$$

This cost function leads to a simple "weight decay" term in the weight update equation (Equation 4.14):

$$w_k(t+1) = w_k(t) - \varepsilon \frac{\partial E}{\partial w_k(t)} - \lambda w_k. \tag{3.21}$$

Here E is the sum of squared errors from Equation 1.3, and the "momentum" term has been omitted. This is easy to implement, and limits the development of any weights with "large" absolute value. The trade-off with the minimization of E is determined by the parameter λ. The argument for using this cost function is that it prevents the network from "over-fitting" the data, since weights that are unimportant for the task (and consequently have $\frac{\partial E}{\partial w_k} \approx 0$) will decay to zero at a rate determined by λ. Another justification that is claimed for this approach is that it minimizes the sensitivity of the output to noise in the input [Nowlan and Hinton, 1992a], although this claim is based on results derived for a linear system. This approach can also be interpreted, from a Bayesian perspective, as specifying a particular prior probability distribution of the weights. This is discussed below as a special case of "Soft Weight-Sharing".

3.3.2 Soft-Weight Sharing

Soft Weight-Sharing [Nowlan and Hinton, 1992a; Nowlan and Hinton, 1992b] is, in essence, a generalization of the Weight Decay approach. Rather than introducing an auxiliary function that causes all weights to have a tendency to decay to zero, they model the weight space as being the result of having selected each weight from a population having a prior probability distribution given by a mixture of Gaussian probability distributions, with nonzero means. The complexity measure is then based on how probable the current state of the network is, given this prior distribution. The aim of this technique is to cause the weights in the network to cluster in "families" close to the means of the Gaussians. Weight decay can be viewed as the result of this technique applied using a prior distribution consisting of a single zero-mean Gaussian with variance $\frac{1}{\lambda}$. The means, variances and mixing proportions of the Gaussians are adjusted simultaneously with the weights during training.

Like weight decay, Soft Weight-Sharing is based on assumptions about the desired properties of the weights that are not derived from the task at hand, and consequently is also vulnerable to small training sets.

3.3.3 Tangent Prop

The aim of Tangent Prop [Simard, Victorri, Cun and Denker, 1992] is to incorporate *a priori* knowledge of the desired invariances of a network into the training procedure. If the response of the network is invariant under some distortion (*e.g.*, rotation, translation, dilation) of the input pattern, then the derivative of its output with respect to such distortions should be zero. For a truly invariant system, this derivative would vanish everywhere. Tangent Prop seeks neural networks invariant under *small* spatial distortions of the training patterns. It thus seeks weight configurations for which the derivative vanishes *for the given training examples*. This approach can lead to "local" invariance (in the neighbourhood of the training patterns), but can only produce global variance by chance.

When a pattern P is transformed by a transformation s depending on n parameters, the set of all possible transformed patterns $S(P)$ is a manifold of at most n dimensions. The patterns in $S(P)$ obtained from *small* transformations of P (*i.e.* the part of $S(P)$ close to P) can be approximated by the tangent plane to the manifold $S(P)$ at point P. Small transformations of P can be obtained by adding to P a linear combination of the vectors that span the tangent plane.

Tangent Prop approximates the vectors corresponding to such small transformations by taking the finite difference between actual training patterns and distorted versions computed from them. The usual least squares error function is augmented with a regularizing term E_r that is proportional to the square of the error of the derivatives

of the outputs with respect to the known desired invariances. The weight-update rule becomes:

$$\Delta w_{ij} = -\varepsilon \frac{\partial}{\partial w_{ij}}(E + \lambda E_r), \tag{3.22}$$

where, for an input pattern x, network output $G(x)$ and a transformation operator parameterized by α, $s_i(\alpha, x)$,

$$\begin{aligned} E_r &= \sum_{x \in \text{training set}} E_r(x) \\ E_r(x) &= \sum_i \left\| K_i(x) - \left(\frac{\partial G(s_i(\alpha, x))}{\partial \alpha} \right)_{\alpha=0} \right\|^2, \end{aligned} \tag{3.23}$$

where $K_i(x)$ is the desired directional derivative of G in the direction induced by transformation s_i applied to pattern x. For the case of *local invariance*, $K_i(x)$ is simply set to zero. The directional derivative can be written

$$\left. \frac{\partial G(s_i(\alpha, x))}{\partial \alpha} \right|_{\alpha=0} = J_G(x) \left. \frac{\partial s_i(\alpha, x)}{\partial \alpha} \right|_{\alpha=0}, \tag{3.24}$$

where $J_G(x)$ is the Jacobian of the network transformation G. This directional derivative can be computed by a forward propagation of the estimated tangent vector through a linearized version of the network.

Training a network using Tangent Prop is thus similar to using plain backpropagation, with the addition that tangent vectors are also propagated through the network. The technique provides a means of introducing *a priori* known desired invariances directly as a constraint on the training of the network, but can only achieve invariance in the neighbourhood of its training patterns. It still requires a large training set if invariance is desired over an entire input space.

Leen (1995) makes explicit the relationship between the use of the regularizer proposed in Tangent Prop and simply augmenting the training set with the distorted versions of the input patterns used to compute the tangent vectors. He shows that learning is equivalent if the weighting term λ in Equation 3.22 is chosen to be the variance of the distortions introduced into the original training set.[5] This shows that regularization in this form really belongs to the family of large training set methods, rather than being a true constraint on the network architecture or weight space.

[5]The equivalence only holds up to order $\mathcal{O}(\sigma^2)$. This is reasonable since Tangent Prop's approximations only hold for small distortions.

3.4 Invariant Representations and Features

3.4.1 Invariant Representations

Associative Memory and ART techniques

In this thesis, many of the Model-Based Neural Networks proposed consist of a neural system which pre-processes the input pattern to extract invariant features, or transforms the input pattern into a representation which is invariant under certain transformations. Such networks have been proposed before, although usually restricted to very simple transformations and input patterns. Several are described below.

Associative Memories

The generalization properties of associative memories have been discussed in §3.1.1. Some attempts have been made to construct associative memories that can correctly associate transformed versions of the input patterns with which they were trained. Kree and Zippelius (1988) propose a network that can recognize topologically-equivalent graphs. The system consists of two coupled networks: a Hopfield network to store and retrieve that data, and a preprocessor to transform the input data. Graph-matching is performed by generating isomorphisms of the input graph and using these to try to retrieve a learnt graph. The use of a network that transforms the input data to an invariant representation makes this a MBNN in the sense of this thesis.

Another scheme is a neural network implementation of cross-correlation [Austin, 1989b; Austin, 1989a]. The input patterns used are n-dimensional binary vectors: a simple straight line of 1s and 0s. A preprocessing system is used which computes all possible shifts of the linear input pattern, and computes the Hamming distance between these and the stored patterns. The minimum Hamming distance indicates the best match. It is shown how a more efficient pattern encoding scheme can reduce storage requirements. Importantly for this thesis, Austin shows how this may be implemented in a neural network, making this a MBNN in our sense, albeit for a very simple problem. Since this network computes all possible transformed versions of the input pattern, it is in fact doing a form of cross-correlation.

Adaptive Resonance Theory (ART) Networks

Another system which uses a pre-processing network to produce an invariant representation of its input pattern is proposed by Srinivasa and Jouaneh (1992). Here the classification stage used is an ART network [Carpenter and Grossberg, 1987; Carpenter and Grossberg, 1988]. The invariance net is designed to be invariant under translations and 90° rotations of an 8×8 input layer. The invariance net thus consisted of 256 "slabs" of 8×8 neurons each, each slab corresponding a a particular combination of

shift and rotation parameters. This approach is again just multidimensional cross-correlation, and is extremely expensive in both space and computation. Each slab is connected with equal weights to the classifier. Since all possible allowed transformations of the patterns are computed for the input pattern, this 16384-dimensional representation of the 64-dimension input pattern is indeed invariant, though far from efficient. This system is shown to be able to classify correctly transformed versions of 4 uppercase alphabetic characters. A version of this system that has better noise tolerance is given in [Srinivasa and Jouaneh, 1993].

3.4.2 Invariant Features

The simplest way to do invariant pattern recognition with a neural network is to present the network not with the input pattern itself, but with an invariant representation of the pattern computed by some other system. Training and recognition take place entirely with the invariant representation, and the neural network is used only as a classifier. Any of the invariant pattern representations discussed in Chapter 2 could be used in this way.

Some studies have been done which investigate the performance of neural network classifiers for such data. For example, Khotanzad and Lu (1990) compare MLPs trained with backpropagation trained with geometrical moments (see §2.2.1) and Zernike moments (see §2.2.2). They found that performance with Zernike moment features was superior, and in particular more robust to noise. Since the invariance in this approach does not arise from the networks themselves, it is not considered at greater length here.

3.4.3 Higher-Order Neural Networks

An important class of neural networks in the context of this thesis are Higher-Order Neural Networks (HONNS). HONNs can be considered to be a form of MBNN specifically designed for invariant pattern recognition. They differ from any of the MBNNs introduced in this thesis, but they share the same goal: to create a neural network architecture which gives a response with guaranteed invariance properties. Several forms of HONN exist which address various combinations of shift, rotation and scale invariance [Perantonis and Lisboa, 1992; Redding, Kowalczyk and Downs, 1993; Schmidt and Davis, 1993; Spirkovska and Reid, 1994; Delopoulos, Tirakis and Kollias, 1994]. Although these networks achieve some success on this task, they have several severe disadvantages, most importantly the storage requirements and scalability of the approach.

The layer immediately following the input layer of a HONN contains nodes which compute weighted sums of *products* of the activation of the input nodes. For a third

order network, the output of a hidden node y_i is

$$y_i = f\left(\sum_j \sum_k \sum_l w_{ijk} x_j x_k x_l\right), \tag{3.25}$$

where f is the transfer function, and $\{x_j, x_k, x_l\}$ are the activations of three input nodes. Invariance is introduced into HONNs by choosing the weights for pairs (2nd order HONNs) or triples (3rd order HONNs) of input nodes with a given geometrical relationship to be the same. Shift invariance could be obtained with a 2nd order HONN by ensuring that all pairs of nodes joined by a line of a given length and orientation have the same weight. Shift, rotation and scale invariance can be obtained with a 3rd order HONN by ensuring that all triples of nodes corresponding to similar triangles have the same weight.

The most significant problem is the combinatorial explosion of the number of weights required. Input layers for pattern recognition networks typically have hundreds of nodes. A hidden node in a 3rd order HONN would thus have to sum $\binom{100}{3} = 161,700$ product terms. For the optical character recognition problem addressed in Chapter 7, which uses a 56×57 node input layer, each hidden node would have 1.08×10^{10} weights. Not only must these weights be stored, but all of the corresponding triples of input nodes must be searched to find those sharing the required geometrical relationships. This is clearly prohibitive.

There are other problems relating to sensitivity to noise, and sampling problems. On discrete lattices, for instance, there are many more possible similar triangles at large scales than at small ones.

It is not surprising that applications of HONNs do not usually use them in this raw form. Perantonis and Lisboa (1992) present a technique for reducing the required number of weights for third order HONNs based on "approximately similar triangles", defined by angles falling within certain specified tolerances. This allows triangles to be coarsely binned into approximately similar classes, but still requires approximately 10^7 weights for a 400 pixel input layer. This is further restricted by only storing values for triangles with a vertex at a certain point: other triangles are translated there as needed (not a neural operation). They found that this system performed better than a system using Zernike moment features for the recognition of typed computer-transformed numerals, both with and without noise. The best test set performance for patterns which had been both rotated and scaled was 79% correct for the HONN and 63% correct using Zernike moments.

Spirkovska and Reid (1994) investigate several other schemes for reducing the dimensionality of the HONN weight space. The consider local connectivity, where triples are only connected if all the inter-pixel distances fall below some bound, this sacrificing

a degree of scale invariance. Other options include sampled connectivity and regional connectivity. They finally settle upon a coarse coding scheme, where overlaying fields of coarse pixels are used to represent an input field composed of smaller pixels. For example, a 10×10 input layer may be represented without loss of information by 4 layers with 5×5 pixels of twice the size, each offset by one (small) pixel horizontally or vertically. The advantage is that $\binom{10}{3} = 120$, whereas $4\binom{5}{3} = 40$. The HONN is constructed using triples taken from the coarse layers. At larger input layer sizes the difference is even more dramatic.

Chapter 4

Model-Based Neural Networks

4.1 The Model-Based Classifier

In this chapter an implementation of MBNNs is introduced in which the networks are modeled in two senses. First, the multilayer feed-forward structure of TNNs (Traditional Neural Networks, without modeling the connections) is generalized to a multilevel feed-forward structure, in which there are one or more layers of nodes at each level. An example of such a structure appears in Figure 4.2. This allows a level to consist of several layers that act as filters, each with different parameters, that all receive the same input from a layer above. The outputs of these filter layers might then form the input to a conventional classifier.

The second sense in which these networks are modeled concerns the specification of the weights on the connections between individual layers. These MBNNs are to be constructed so that their structures model the tasks which they are to perform. One powerful means of achieving this is to make the weights of the connections between nodes depend on the relative positions of the nodes. Consequently, the input to a node can be constrained to be a given function of the outputs of its source nodes, taking into account their geometric relationships. The use of the same parameters to specify the weighting functions for many different connections allows the dimensionality of the parameter space, which must be searched during training, to be reduced greatly.

We have introduced a definition of "nodal distance" for each layer of the NN in order to structure information processing in terms of the relative positions of nodes within each layer. In this study, all nodes are assigned Cartesian coordinates within their layers, such that the distance between horizontally or vertically adjacent nodes is 1. The origin of coordinates for each layer is situated at the centroid of the layer. The distance d_{ij} between nodes i and j in different layers is calculated by projecting the layers onto each other so that their centroids coincide and then calculating the distance between nodes using the chosen metric of their coordinates as if they were in the same

layer. In this sense, the topology of the layers and the metric used play an important role in this form of MBNN. In the work that follows, the metric used was the 2-norm, the usual Euclidean distance.

We have considered three different forms of connection model. For each destination layer (hidden or not), the weighting function f is defined by one of the following forms.

M.1 Each individual connection has it's own parameters $\mathbf{a}_{ij}$ such that:

$$w_{ij} = f\left(\mathbf{a}_{ij}, d_{ij}\right). \tag{4.1}$$

M.2 For a given destination plane, or set of nodes, each node has new parameters $\mathbf{a}_j$ such that:

$$w_{ij} = f\left(\mathbf{a}_j, d_{ij}\right). \tag{4.2}$$

M.3 Each node of a given destination plane has the same parameter values:

$$w_{ij} = f\left(\mathbf{a}, d_{ij}\right). \tag{4.3}$$

The backpropagation training procedure can be extended using the Chain Rule so that the gradient descent is performed on the weighting function parameters $\{\mathbf{a}_{ij}\}$, rather than on the weights themselves. In all three cases the Chain Rule applies in the same way. One extra step in differentiation is required (see Equation 4.11), so the degree of computational complexity increases. Indeed, it should be noted that there is a trade-off between computational complexity in terms of the number of differentiations required and the number of parameters, in such situations. However, we will show that the parameter reduction of the MBNN is significant enough to compensate for this increase in complexity. Further, we will show that such significantly lower-order representations prove to have greater robustness to noise, and generalize with respect to the invariance characteristics of the network geometry, connection model and class sample variabilities.

As with traditional NNs, the Chain Rule is used to relate the shape of the transducer function to the error function (Equation 1.3). That is,

$$\frac{\partial E}{\partial y_i} = \sum_c \left(y_{jc} - t_{jc}\right) \tag{4.4}$$

and

$$\frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial x_j}. \tag{4.5}$$

For our first implementation of a MBNN, we have modeled connection weights $\{w_{ij}\}$

as functions of a vector of parameters, $\mathbf{a}_{ij}$, and the distance d_{ij} between the nodes:

$$w_{ij} = f\left(\mathbf{a}_{ij}, d_{ij}\right). \tag{4.6}$$

In fact, the usual TNN model is a degenerate case of Equation 4.6, where

$$w_{ij} = a_{ij}. \tag{4.7}$$

For this application, three potentially useful weighting functions are: one, a Gaussian:

$$w_{ij} = a_1 \exp\left(-a_2 d_{ij}^2\right); \tag{4.8}$$

two, the Gabor function:

$$w_{ij} = a_1 \exp\left(-a_2 d_{ij}^2\right) \cos\left(a_3 x + a_4 y + a_5\right); \tag{4.9}$$

and three, a function implementing a radially-symmetric narrow-band filter:

$$w_{ij} = \cos\left(a_1 d_{ij}\right). \tag{4.10}$$

Using the Chain Rule, we obtain the equivalent derivative of the error function, now with respect to the parameter a_k, defined by:

$$\frac{\partial E}{\partial a_k} = \frac{\partial E}{\partial w_{ij}} \frac{\partial w_{ij}}{\partial a_k}. \tag{4.11}$$

When a parameter is shared by more than one connection, it is necessary to sum the partial derivatives of the error with respect to that parameter over all the connections that depend on the parameter.

For the TNN case, $\frac{\partial w_{ij}}{\partial a_k} = 1$ for all i, j, k. For the Gaussian case, for example, we obtain:

$$\frac{\partial w_{ij}}{\partial a_1} = \exp\left(-a_2 d_{ij}^2\right) \tag{4.12}$$

and

$$\frac{\partial w_{ij}}{\partial a_2} = -a_1 d_{ij}^2 \exp\left(-a_2 d_{ij}^2\right). \tag{4.13}$$

Further, the gradient descent methods work in exactly the same way as for the non-parametric representations, resulting in the iterative state space equation:

$$a_k(t+1) = a_k(t) - \varepsilon \frac{\partial E}{\partial a_k(t)} - \alpha \frac{\partial E}{\partial a_k(t-1)} \tag{4.14}$$

where ε and α are set by the user to determine the resolution of the convergence or search procedure. Typical values are $\varepsilon = 0.005$ and $\alpha = 0.5$.

4.1.1 Relation to Adaptive Filtering

For this specific formulation, the MBNN essentially becomes a generalization of the well-known filter estimation problem in Adaptive Filter Theory. Specifically, the three cases discussed above (M.1, M.2 and M.3) fit in with filter estimation in the following ways. The discrete convolution function, defining the action of a filter $h(x, y)$ on a signal $I(x, y)$ (here, two-dimensional), is defined by:

$$R(x, y) = \sum_{uv} h(u, v) I(x + u, y + v). \tag{4.15}$$

The output coordinate system in Equation 4.15 corresponds to the destination layer in the TNN. That is, the filter kernel h acts as a weighting function (w, or mapping) in the form of a moving window over the input signal layer (see Figure 1.1). Let us now make a distinction between the coordinates in the input and output image planes. Let (x_0, y_0) and (x_1, y_1) be the input and output coordinates respectively. Case M.1 corresponds to the highest form of non-stationarity, where the parameters $\mathbf{a}$ of the kernel are a function of both the input and output position pairs:

$$w(u, v) \equiv w(u, v, \mathbf{a}(x_0, y_0; x_1, y_1)); \tag{4.16}$$

case M.2 corresponds to:

$$w(u, v) \equiv w(u, v, \mathbf{a}(x_1, y_1)); \tag{4.17}$$

and, finally, case M.3 corresponds to:

$$w(u, v) \equiv w(u, v, \mathbf{a}). \tag{4.18}$$

Case M.2 corresponds to a generalized Wigner function [Wechsler, 1990] when both input and output coordinate planes are identical; and case M.3 corresponds to the usual shift invariant filter in so far as the kernel is constant over all positions in the input and output layers.

From this perspective the NN provides a more general structure than the usual adaptive filter formulation where the filter function parameters are assumed to be shift-invariant and do not involve additional nonlinear transducer functions. It should be noted, at this stage, that a number of past implementations have used adaptive filters as preprocessing units for NNs [*e.g.* Wechsler, 1990]. However, the aim of this investigation is to study how such structures, in their most general forms, can be formulated in the

same architecture as the remainder of the NN.

4.2 A Simple Invariance Example

The correct classification of patterns that have been transformed in some way (shifted, rotated, or scaled) is an inherently difficult one for TNNs. Since all connections in a TNN have independent weights, the output of such a network inevitably depends upon the activation of *specific* nodes in the input layer. It cannot be expected that a TNN's response to an asymmetrical input pattern that has been rotated (for example) will even be related to its response to the initial pattern. Moreover, since the training procedure is unconstrained, it is possible that correct classification of a training set could be achieved on the basis of the chance activation of a single node in all the examples of one class in the training data. Such a "solution" totally fails to capture the invariant features necessary to discriminate between the classes in general.

As discussed in §3.2.1, a typical TNN approach to the invariance problem is to include shifted, scaled or rotated versions of patterns in the training set. Given a sufficiently large and varied training set, and enough degrees of freedom in the network (*i.e.*, enough independent weights), a solution can be attained by this method. This seems, however, to be an inappropriate approach to extracting salient features. Rather than implementing "real" invariant pattern recognition, the network has been presented with a large number of essentially distinct patterns that *inevitably* activate it in different ways, and then trained to label them as members of the same class. The invariance is a consequence of the properties of the training data, rather than being an inherent property of the network itself.

Tangent Prop (see §3.3.3) differs from this approach in that it augments the usual least squares cost function with a regularizing term that depends on the magnitude of the derivative of the outputs of the network with respect to various distortion operators. If the network has a derivative of zero with respect to a given distortion operator, then it is invariant with respect to small distortions of the input image corresponding to the action of that operator. Tangent Prop is thus close to MBNNs in its aim: to use *a priori* knowledge of the desired invariances of the network directly, rather than requiring a training set that is sufficiently varied to cause such invariances to arise spontaneously. It does not, however, place any explicit constraints on the weight space to be searched.

Using a MBNN, it is possible to *intrinsically* encode invariant features in the structure of the network and weight space. To demonstrate this, we have chosen the following simple example. The task is to distinguish between high and low frequency regular textures presented as patterns at an input layer consisting of 15×15 nodes, regardless of their orientation. It is desired, however, to train the network using only textures of a single orientation. Consequently, the network's response must be invariant under

rotations and shifts of patterns on the input layer.

The MBNN used consisted of a 15×15 input layer, connected using M.3 radially-symmetric cosine weighting functions (Equation 4.10) to two 15×15 hidden layers, which were, in turn, connected using M.2 simple weights to a 2×2 hidden layer. This hidden layer was then connected using M.1 simple weights to a 1×2 output layer. This configuration has only $1+1+2\times4+2\times4 = 18$ independent parameters. The fact that information from the input layer only enters the network by passing through radially-symmetric weighting functions ensures that the network's response will be invariant under rotations of the input pattern, in place. The M.2 connections to the next layer simply implement a summation of the outputs of each of the two layers above, thus making the system invariant under shifts of the input pattern. It is clear that much information contained in the input pattern is lost in this process, but that is just the aim of classification – the removal of variance irrelevant to the discrimination of the classes in question. The performance of this MBNN was compared to that of a TNN having a 15×15 input layer, 2×2 hidden layer, and a 1×2 output layer. This TNN has 908 independent parameters.

The training and test patterns used in this simulation were sampled versions of a two-dimensional regular texture. The activation $I(x, y)$ at node (x, y) of the input layer was given by:

$$I(x,y) = \frac{1}{2}K\left(1+\cos\left(2\pi\left(u\frac{x}{x_{\max}}+v\frac{y}{y_{\max}}\right)\right)\right). \tag{4.19}$$

Here $x_{\max}$ and $y_{\max}$ are the horizontal and vertical dimensions of the input layer in nodes, u and v are the horizontal and vertical components of the spatial frequency in cycles per layer, and K is a normalizing factor introduced so that the total activation summed over all input nodes (x, y) was equal for all combinations of u and v, except for quantization errors. K is found by integrating over the input layer with respect to x and y:

$$K = \left(1 - \frac{\cos\left(2\pi\left(u+v\right)\right)+1-\cos\left(2\pi u\right)-\cos\left(2\pi v\right)}{4\pi^2 uv}\right). \tag{4.20}$$

Note that this correction is necessary only when either u or v is not an integer. The reason for using normalized cosine gratings rather than just bands of 1's on a background of 0's is so that the total (sum) activation of the input layer is the same for both high and low frequency input patterns. This prevents the TNN achieving "pseudo-rotation invariant" performance by simply "counting ones" on the input layer.

Given that the radial frequency of the gratings is defined by:

$$f = \sqrt{u^2+v^2}, \tag{4.21}$$

the training set consisted of 10 patterns, all with horizontal textures ($u = f, v = 0$). Five of the patterns were "low frequency", with $f = 1.00$, 1.25, 1.50, 1.75 and 2.00 cycles per layer, and five were "high frequency" with $f = 6.00$, 6.25, 6.50, 6.75 and 7.00 cycles per layer. The test set consisted of 20 patterns: 10 with diagonal textures, and 10 with vertical textures. The f values were identical to those of the training set.

Five of the TNNs and five of the MBNNs described above were prepared for this problem. The parameters of the TNNs were initially set to values randomly selected from the range $[-W, W]$. The value of W used for the input weights to a given node was chosen using the following formula:

$$W = \frac{5}{\text{number of inputs to node}}. \tag{4.22}$$

This ensures that the input to the sigmoid nonlinearity is in the range $[-5, 5]$, rather than in the flat "on" or "off" regions where the derivative of the sigmoid is zero, which renders backpropagation unable to affect the input weights. Other optimization techniques have trouble with such "saturated" nodes also, since small changes in the input weights cause no appreciable change in the output.[1]

For the MBNNs, the pass frequencies of the two cosine filters were both initially set to 3.5 cycles per layer (the centre of the Nyquist range), and all the simple weights were initialized to zero.

These networks were not, in fact, trained using backpropagation. The reason for this is the manifest nonmonotonicity of the error surface of the MBNNs. Since the weights to the first two hidden layers are generated by cosine functions, local minima abound. Using backpropagation, it is not possible for the pass frequencies to be adjusted to the appropriate values, since the derivative with respect to this parameter has a sinusoidal component. Gradient descent is consequently an inappropriate optimization technique for this problem. Whilst this is to some extent a disadvantage, it must be remembered that local minima are a problem, often left unmentioned, for TNNs also.

The optimization technique used was simulated annealing [Metropolis et al., 1953]. The following cost function was used for the network's performance on the training patterns:

$$c = \frac{\lambda \frac{1}{2} \left(\frac{\sinh(\kappa(1-\Psi))}{\sinh(\kappa)} + 1 \right) + \frac{\sum_j \sum_c (y_{jc} - t_{jc})^2}{jc}}{1 + \lambda}. \tag{4.23}$$

The variable Ψ is the fraction of the training patterns classified correctly. The first term of this cost function is thus an explicit measure of classification performance. The

[1] Note that this initialization method was used for generating all weights for networks trained using backpropagation throughout this thesis.

sinh function is used so that classification is rewarded nonlinearly. The function:

$$\frac{1}{2}\left(\frac{\sinh\left(\kappa\left(1-\Psi\right)\right)}{\sinh\left(\kappa\right)}+1\right) \tag{4.24}$$

varies between 0 and 1 as Ψ varies between 1 and 0. It has the values 0.5 when Ψ equals 0.5, and is reasonable "flat" for values of Ψ around 0.5. This means that small variations around 50% correct classification have little effect on the cost, but very poor performance is heavily penalized and very good performance is greatly rewarded. The parameter κ determines the degree of nonlinearity of this function. For $\kappa = 1$, the function is very nearly linear on the specified domain. The second term is just the squared error (as in Equation 1.3) normalized to a range of $[0, 1]$. The parameter λ determines the trade-off between these two components of the cost. For these simulations, the values $\kappa = 5$ and $\lambda = 10$ were used. The entire cost function is normalized to the range $[0, 1]$.

This cost function has the advantage that the notion of classification is explicitly included. Using the squared error (right-hand component of Equation 4.23) alone can result in degraded classification performance, since a large improvement in the error on one training pattern can compensate for small changes in the error on more than one other pattern, even if these small changes result in incorrect classification. The first term in the cost function serves to counter this effect. The squared error term is retained so that the cost function does not have large "plateaus" in the regions where classification performance is constant. This allows change in parameters which reduce the squared error to be accepted, even if they do not affect classification performance.

The networks were trained using the simulated annealing algorithm. Each of the parameters of the network was perturbed by a value randomly selected from the range $[-0.5, 0.5]$ at each iteration, and the cost function C evaluated. The usual simulated annealing criterion was then used to determine whether or not the perturbed parameters were to be accepted. It was found that a low "temperature" parameter was best for this task. In fact, the results reported here are for networks trained with a temperature parameter of zero. This means that any perturbation of the network parameters that resulted in an increase in C was rejected. This does not make the procedure equivalent to gradient descent, however, since the parameters are perturbed randomly. The ability of the system to escape local minima depends upon the size of the random perturbations compared to the depths of the local minima.

Each TNN was trained using this technique for 1000 iterations. The MBNNs were trained for 2000 iterations. This difference may be attributed to differing error surfaces, due to the different architectures, and also to the different search procedures used. They are thus not directly comparable. It should be noted the 100% correct classification of the training patterns was usually achieved in far fewer iterations (average 48 iterations

for the TNNs and 195 iterations for the MBNNs). The results appear in Table 4.1.[2]

TNN (1000 iterations)		MBNN (2000 iterations)	
Training	Test	Training	Test
100	45	100	100
100	60	100	100
100	70	100	85
100	45	100	100
100	45	100	100
$\mu \pm \sigma$	53 ± 12	$\mu \pm \sigma$	97 ± 7.0

Table 4.1: Classification performance (percent correct) of traditional and Model-Based Neural Networks on 10 training patterns and 20 test patterns after training using stochastic relaxation.

It is clear that the MBNNs, as expected, generalize to the rotated test patterns extremely well. In fact, in most cases, they generalize perfectly. The TNNs, on the other hand, to within one standard deviation, do not generalize at all, since 50% correct classification corresponds to randomly labeling all input patterns as belonging to one of the two classes. Moreover, this greatly improved performance was achieved by a MBNN specified by only 18 parameters, compared to a TNN with 908 parameters. That is a 50-fold reduction in the size of the parameter space. Note that the minimum size TNN for this task (if it was solved as a two-layer problem) would still have 450 parameters. This example clearly demonstrates that the use of MBNNs allows desired invariances to be specified *a priori*, as well as achieving a significant reduction in the dimensionality of the state space to be searched. The use of a stochastic relaxation training technique (simulated annealing) allows the MBNNs to be trained, despite the fact that they have complicated, nonmonotonic error surfaces that make solution using backpropagation infeasible.

4.3 The Plaut and Hinton Study

To illustrate further the differences between this model and more traditional ones, we have considered the problem of classifying "rising" and "non-rising" signals in speech spectrograms, as studied by Plaut and Hinton (1987). This task is an excellent example of a problem for which an expert's knowledge of the important features for the task can be used to construct a MBNN that is constrained to classify on the basis of those features, and not on irrelevant differences between training patterns. The task consists

[2]Throughout this thesis, training and test set results are presented for multiple copies of identical networks with differing random weight initializations, so that the repeatability of the results may be judged. Throughout μ is the mean of the classification performance, and σ is the standard deviation.

of classifying as risers or non-risers synthetic spectrograms, represented as patterns on a 6×9 grid, where the vertical direction corresponds to frequency, and the horizontal direction to sampling time. The patterns are generated under the following constraints:

1. Generate Risers (R) and Non-risers (NR) with equal probability.
2. Randomly choose one of the six possible frequencies for Non-risers. Randomly choose one of the four highest frequencies for Risers.
3. Pick one of five possible onset times at random.
4. Assign the value 0.4 to each unit cell that is part of the signal, and 0.1 to the background elements.
5. Add pseudo-Gaussian white noise to each element.

Examples of these signals are shown in Figure 4.1.

This procedure generates a population (neglecting noise cases) with 50 members: 20 risers and 30 non-risers. Plaut and Hinton (1987) attacked this problem using a traditional NN with one hidden layer. Their network consisted of 54 input layer units, 24 hidden layer units, and, of course, 2 output layer units (R, NR) – see Figure 1.1.

Using backpropagation, they trained this network using 10,000 blocks of 25 examples, each example being generated randomly when required. By the end of this process, the network was able to classify correctly 97.8% of unseen patterns. It must be remembered, however, that this is in the context of a parameter space with $54 \times 24 + 24 \times 2 = 1344$ dimensions, and an enormous set of training examples, completely spanning the population.

4.3.1 A MBNN solution to the Riser/Non-riser Problem

We have implemented their procedure and compared it to our model-based system. The MBNN used consisted of a 6×9 input layer, connected to two 6×9 hidden layers at the second level using M.3 (see Equation 4.3) and Gabor weight functions (see Equation 4.9), so that these layers were implementations of Gabor filters responding to line elements of given orientations in the input layer. These two Gabor layers were connected to a conventional hidden layer of nine units, using M.1 connections and this, using M.1, was connected to two output layer units. This network is specified by $5 + 5 + 54 \times 9 + 54 \times 9 + 9 \times 2 = 1000$ parameters, a reduction of 25.6%. It should be pointed out that the critical factor in determining this figure is the size of the third layer. Smaller layers were not tried, but should, intuitively, work. This system is shown in Figure 4.2.

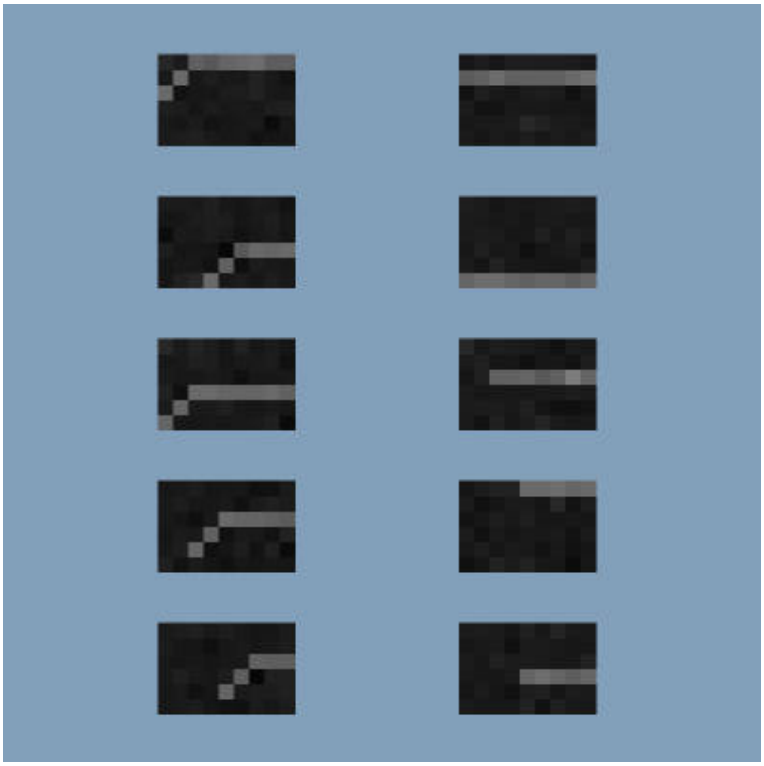


Figure 4.1: Examples of Risers (left) and Non-risers (right) from the Plaut and Hinton (1987) classification problem. Signals were to be classified as R or NR as a function of their spectrogram shapes. These are the training patterns used in the simulations in this thesis.

A rectifying transfer function of the form:

$$y = \frac{1 - e^{-x^2}}{1 + e^{-x^2}} \tag{4.25}$$

was used for the outputs of the units in the Gabor layers, so that all nonzero responses were counted as evidence for a feature being present (evidence for a particular orientation response). All other transducer functions were of the usual sigmoid form, as shown in Equation 1.2.

The initial values of the two parameters specifying the orientations of each of the Gabor filters were set so that one filter would respond to diagonal elements, and the other to horizontal elements. This preselection process may be justified by the fact that, if an initial pre-backpropagation stage of training were included in which successive filter orientations were tried, the two for which the filters' summed outputs were greatest would have resulted in these two filters. Such a "self-organization" process is not, however, the focus of this study.

By explicitly including in the network knowledge of the significance of the presence of lines of given orientation, it was expected that, in addition to reducing the

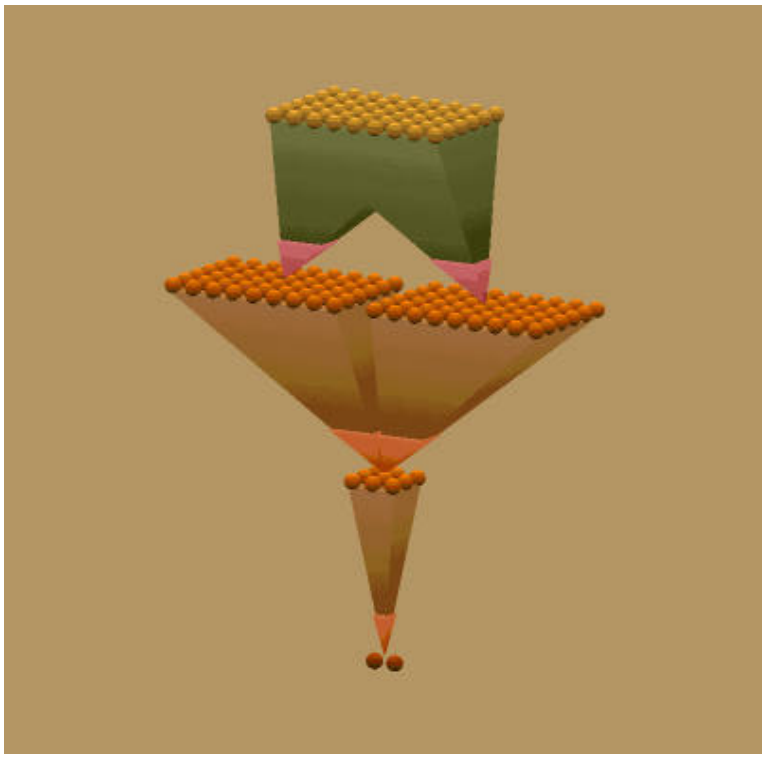


Figure 4.2: The MBNN used to solve the riser/non-riser classification problem in this study. Input (top) to second level connections are modeled by Gabor (M.3)-type connections. Other connections are defined by M.1 connections.

dimensionality of the parameter space, the number of training passes required would be reduced and the need to have a set of training patterns that spanned the population would be removed. That is, the network would generalize to the classification of patterns not present in the training set, since layers responding to the critical feature for classification (line orientation) had been included.

Five of each of these two kinds of networks (with different initial parameter seedings) were then trained, using backpropagation. A fixed set of patterns was used, consisting of five risers and five non-risers, which were initially selected at random from the population described above (see Figure 4.1). The parameters were adjusted after each pass through this set. Also, the performance of the network was measured, both in terms of the percentage of the training set correctly classified, and the percentage correctly classified of a test set, consisting of 100 risers and 100 non-risers not used in the training process.

Both kinds of network achieved 100% correct classification on the training set, as would be expected for such a small set of patterns. The MBNNs achieved this more rapidly than the TNNs, with an average of 1224 passes required, compared to 1809 passes for the TNNs (using the same backpropagation parameters). The major

difference between the two models, however, is their ability to generalize.

The performance of the TNNs in classifying patterns that did not form part of their training set actually worsens as they approach 100% correct classification of their training sets, eventually declining to an average of 51.1% correct after 2000 passes: the TNNs have very little ability to generalize.

The performance of the MBNNs is strikingly different. The average percentage of the test set correctly classified was 88.5%, after 2000 passes. This is comparable to the performance of the Plaut and Hinton (1987) network trained with 250,000 different examples, but achieved with a total of only 10. The need to span the pattern population has been removed, as has the requirement to present patterns with numerous different noise structures. Summaries of the TNN and MBNN performance with new patterns is shown in Table 4.2.

	TNN	MBNN
Network 1	50.5	99.0
Network 2	50.5	74.5
Network 3	51.5	98.0
Network 4	50.5	74.0
Network 5	52.0	97.0
$\mu \pm \sigma$	51.0 ± 0.6	88.5 ± 11.7

Table 4.2: Classification performance (percent correct) of networks on 200 novel patterns after 2000 training iterations on 10 patterns.

Figures 4.3 and 4.4 show the responses of one of the MBNNs to a riser and a non-riser, respectively. The output layer responses show that the two patterns have been classified correctly. Further, the clear complementarity of the penultimate layer responses suggests that the dimensionality of this layer could be significantly reduced. The most interesting feature of the responses, however, is the way in which the training has altered the initial parameters of the Gabor filter layers. The left-hand Gabor layer was initialized to respond weakly to line segments oriented at 45° to the horizontal (with positive gradient), and the right-hand layer to respond weakly to horizontal line segments. The effect of training on the right-hand layer is predictable in so far as that the filter responds more strongly to horizontal segments, with more nodes being activated by a given horizontal line. The behaviour of the left-hand layer is more interesting. The orientation of the filter has been rotated by the training process so that it responds to the *vertical* component of the rising part of the risers. The Gaussian component of the M.3 model (Equation 4.9) has been broadened so that any vertical component present in the input pattern causes nodes to be activated all across the filter layer. By selecting the vertical components of the risers rather than the diagonals, the

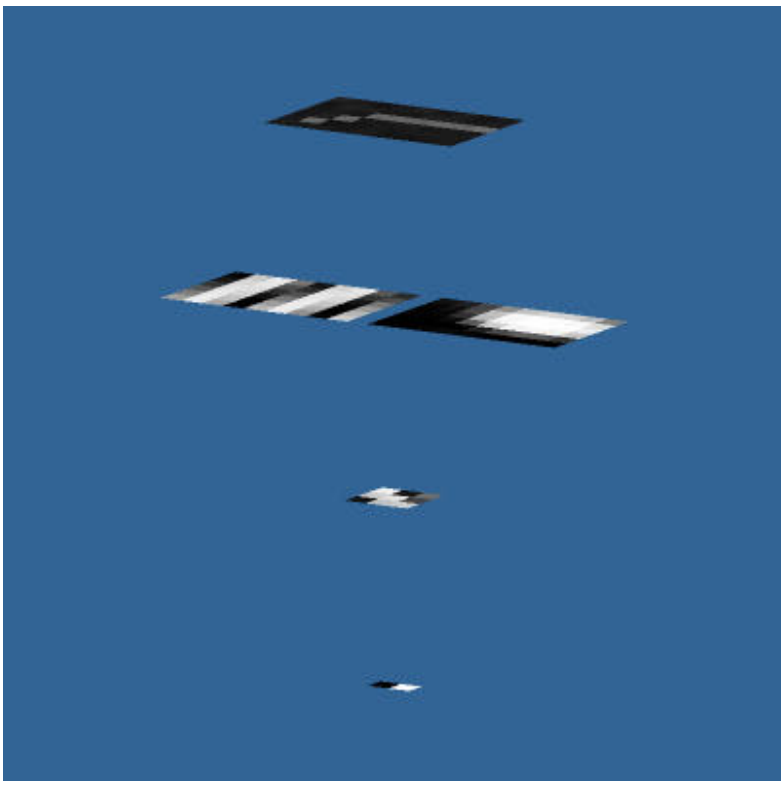


Figure 4.3: Response of MBNN to a riser. Note that the left-hand Gabor layer is responding to the vertical component of the pattern.

filter has "learnt" to respond to that feature of risers that is *orthogonal* to non-risers. This illustrates the analogy between the network's function and principal components analysis (see §1.1).

It is apparent in this simple two-class classification problem that a much lower dimensional solution could have been obtained by using direct adaptive filter theory. However, we have retained the TNN structure to illustrate how our MBNN can fit easily within the TNN architecture.

4.4 Other Techniques For Improving Generalization

As discussed in Chapter 3, most previous efforts to improve the generalization performance of NNs on novel data differ from the MBNN approach in a fundamental way. Only one of the other techniques reviewed here challenges the notion that the training data set alone is sufficient to determine the parameters and their relationships (if any), and in some cases the architecture, of the NN. These methods, therefore, make the tacit assumption that the training set is sufficiently large and varied to characterize the task in general. None of them places restrictions on the parameter space to be searched during training. The MBNN approach, conversely, allows the network, and

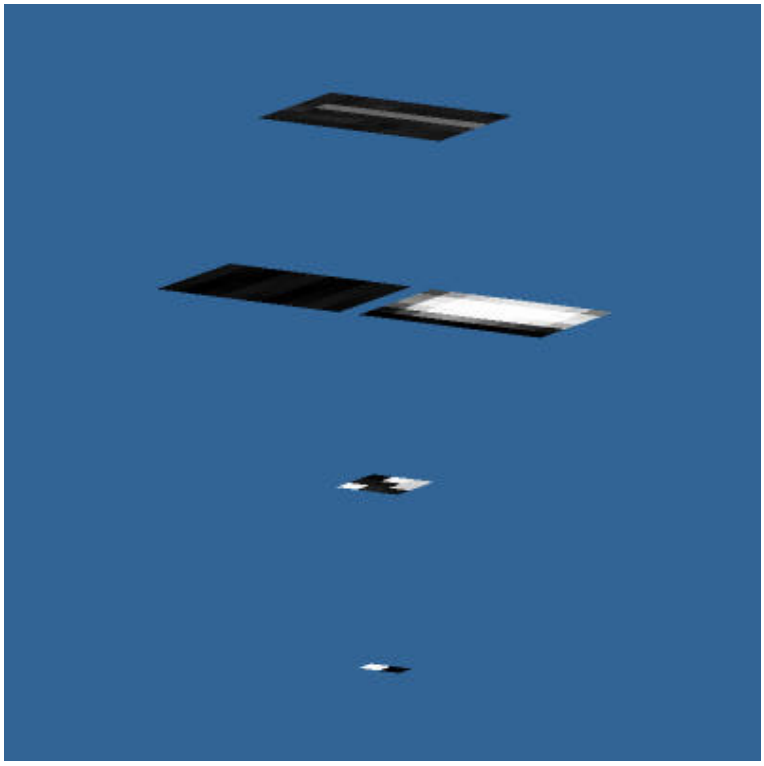


Figure 4.4: Response of MBNN to a non-riser.

its parameters, to be constrained in a way that is known to reflect the properties of the task that it is to perform. One of the aims of the MBNN approach is to remove the need to have a training set that spans the population of possible input patterns, requiring only that the training set spans the population in the feature space determined by the explicit modeling of the network.

A common approach to improving generalization is to use a regularizing term in the cost function. This is usually some measure of the "complexity" of the weights, and is independent of the application, as in Weight Decay (see §3.3.1) and Soft Weight Sharing (see §3.3.2). Conversely, in the MBNN approach, the constraints on the weight-space search are derived by explicitly modeling the task that the network is to perform.

An attempt was made to implement Soft Weight-Sharing so that its generalization performance could be compared to that of MBNNs on the riser/non-risers problem. Repeated efforts failed to achieve a solution using backpropagation. It was suspected that this may have been due to the Soft Weight-Sharing problem being inherently ill-conditioned. This was later confirmed by the author (Nowlan, personal communication, 1992).

Cascade-Correlation, discussed in §3.2.3, is another technique that is claimed to improve generalization. In this technique, not only the connection weights, but the entire architecture of the hidden layer(s) of the network is determined by the training

data. In this sense, Cascade-Correlation can be seen as the antithesis of the MBNN approach.

Tangent Prop is in the same spirit as the MBNN approach. The aim of Tangent Prop is to incorporate *a priori* knowledge of the desired behaviour of the system into the training procedure (see §3.3.3). It is still fundamentally different to the MBNN technique since the weight space searched during training is not constrained, and the structure of the network is not chosen to reflect desired functionality. Invariance is not guaranteed, but must be learnt. It would be interesting to compare Tangent Prop with MBNNs trained on the same data.

In order to demonstrate that the MBNN approach can lead to superior generalization than techniques that take no account of the known properties of the target system, TNNs identical to those used in the previous examples were trained using backpropagation and weight decay. Five networks with different initial parameter seedings were created in each case, and then copies of these networks were made. These sets of identical networks were then trained using various different values of the weight decay parameter λ (See Equation 3.21. This allows the effect of various values of λ to be compared, since the networks had identical starting conditions.

The results for the networks trained for the regular textures problem appear in Table 4.3. To within one standard deviation, there is no difference between the performance of the networks trained with no weight decay ($\lambda = 0$) and those trained with each of the various values of λ. The trend, if any significance can be attached to it, is that weight decay degrades generalization performance for this task. The generalization performance of all these backpropagation-trained TNNs, with and without weight decay, is still well below that of the MBNNs described in §4.2.

	λ			
	0.000	0.001	0.005	0.010
Network 1	65	65	60	65
Network 2	55	70	70	55
Network 3	65	55	50	50
Network 4	65	60	60	60
Network 5	65	60	60	50
$\mu \pm \sigma$	63 ± 4.5	62 ± 5.7	60 ± 7.1	54 ± 6.5

Table 4.3: Classification performance (percent correct) of TNNs on texture test data for various values of the weight decay parameter λ, after 1000 iterations.

The results for the application of weight decay to the TNNs used for the riser/non riser problem appear in Tables 4.4 and 4.5. Again, it was not possible to claim that weight decay improved generalization. When $\lambda = 0.005$, the performance on the train-

ing data improved to 53.0%, but the networks failed to achieve 100% classification of the training set. In fact, with $\lambda = 0.01$, the networks were unable to fit the training data at all. The performance of the MBNNs described in §4.3 is greatly superior.

	λ		
	0.001	0.005	0.010
Network 1	100.0	90.0	50.0
Network 2	100.0	90.0	50.0
Network 3	100.0	90.0	50.0
Network 4	100.0	90.0	50.0
Network 5	100.0	90.0	50.0
$\mu \pm \sigma$	100.0 ± 0.00	90.0 ± 0.00	50.0 ± 0.00

Table 4.4: Classification performance (percent correct) of TNNs on Riser/Non-riser training data for various values of the weight decay parameter λ, after 1000 iterations.

	λ		
	0.001	0.005	0.010
Network 1	50.0	53.5	50.0
Network 2	50.0	53.5	50.0
Network 3	50.0	53.5	50.0
Network 4	50.0	53.5	50.0
Network 5	50.0	53.5	50.0
$\mu \pm \sigma$	50.0 ± 0.00	53.5 ± 0.00	50.0 ± 0.00

Table 4.5: Classification performance (percent correct) of TNNs on Riser/Non-riser test data for various values of the weight decay parameter λ, after 1000 iterations.

These results show that weight decay can adversely affect network performance on the training data, and is no guarantee of improved generalization performance.

It can be seen, therefore, that MBNNs offer a superior means of achieving good generalization performance for tasks which are sufficiently well understood to enable MBNNs that encode the necessary invariant features to be constructed. Moreover, previous methods supposed to improve the generalization performance of NNs are often based on beliefs about the desirable properties of the connection weights that are not necessarily valid, and take no account of the nature of the task for which the network is designed.

4.5 An Extremely Low-dimensional Solution to the R/NR Problem

Since the work described in the previous sections was carried out [Caelli, Squire and Wild, 1993], some further investigations have been done which indicate that MBNN solutions for this problem exist with dramatically fewer parameters than those so far described, and with more consistent performance. An example is the network shown in Figure 4.5.

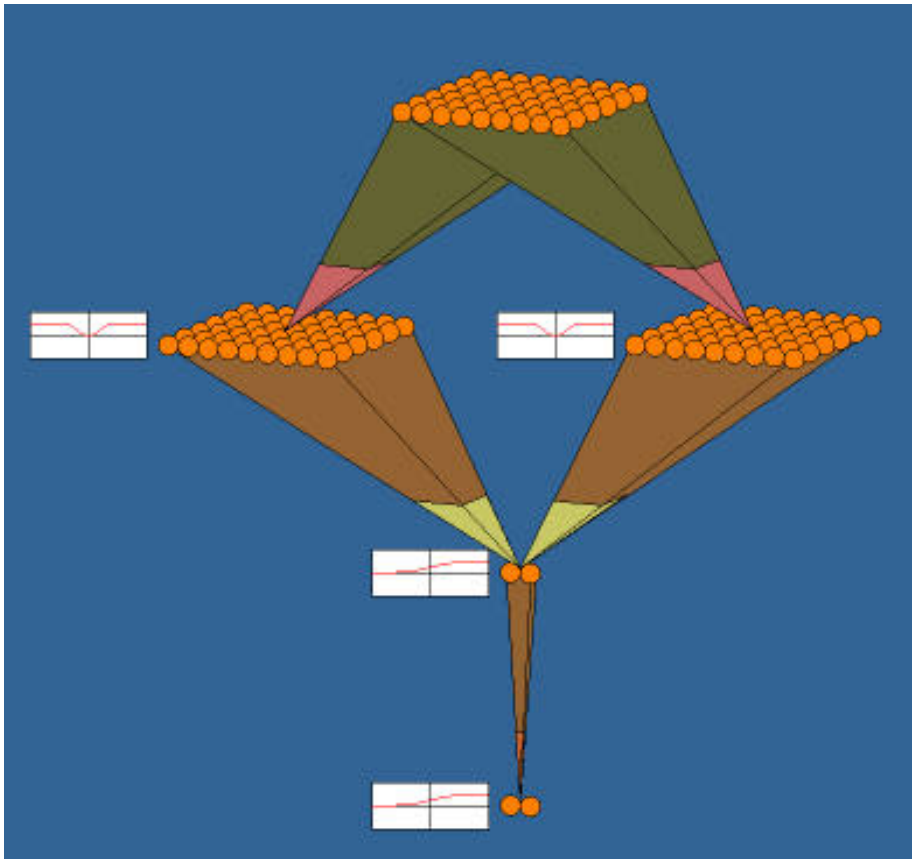


Figure 4.5: A MBNN that solves the Riser/Non-riser problem with only 22 parameters.

This MBNN has a 6×9 input layer, connected to two 6×9 hidden layers at the second level using M.3 and Gabor weight functions, as in the network described in §4.3.1. The transfer function of these layers was the rectifying function given in Equation 4.25. These two Gabor filter layers were connected to a hidden layer of two units, using M.2 simple connections, so that each of these nodes was forced to respond only to the sum energy of the filter layers. These nodes were connected using M.1 simple connections to the two output layer units. This network is specified by $5+5+(2+1)\times2+(2+1)\times2 = 22$ parameters, a reduction of 98.36% over the original network used by Plaut and Hinton (1987). This reduction is possible because an *a priori* design is implemented. Using

expert knowledge of the problem domain, it was decided that the system should consist of the following components:

- Two filters that respond to line orientation, with outputs rectified so that only the "energy" of the response is measured.
- Two nodes that integrate this energy over the entire filter, so that the position of the activation in the layer is irrelevant.
- A simple linear classifier to distinguish between the activation of the two filter integrators.

Note that, as always with MBNNs, all that is left after training is a simple neural network. Importantly, however, although this network has 6056 connections, it has only 22 free parameters during training.

Twenty networks of this architecture were created, each with differently initialized parameters. In this case the initial orientations of the Gabor filter layers were randomly initialized, unlike those in §4.3.1. The training and test sets used in the study described in §4.3.1 were used again. Ten of these networks were trained using simulated annealing [Metropolis et al., 1953; Kirkpatrick et al., 1983], and ten with the extended backpropagation algorithm described in §4.1.

4.5.1 Simulated Annealing

The results for the networks trained using simulated annealing are shown in Table 4.6. These results indicate that solutions which exhibit perfect generalization to the test set

	Best		Final (20000 iterations)	
	Training Data	Test Data	Training Data	Test Data
Network 1	100.00	90.00	100.00	85.00
Network 2	100.00	100.00	100.00	99.50
Network 3	100.00	90.00	100.00	82.00
Network 4	100.00	100.00	100.00	98.50
Network 5	100.00	100.00	100.00	100.00
Network 6	100.00	98.00	100.00	96.50
Network 7	100.00	100.00	100.00	99.50
Network 8	100.00	100.00	100.00	100.00
Network 9	100.00	100.00	100.00	96.00
Network 10	100.00	100.00	100.00	99.50
$\mu \pm \sigma$	100.00± 0.00	97.80 ± 3.95	100.00± 0.00	95.70 ± 6.25

Table 4.6: Classification performance (% correct) of the 22 parameter MBNN classifiers after 20000 iterations of simulated annealing training on the Riser/Non-riser problem.

exist for this 22 parameter network. Moreover, the simulated annealing algorithm is

frequently able to find them, and avoids the catastrophic failures that can occur when using backpropagation on functions with non-monotonic derivatives such as Gabors. It can be seen that a cross-validation algorithm that caused training to stop when 100% classification of the training and test sets was achieved would give excellent results for these networks.

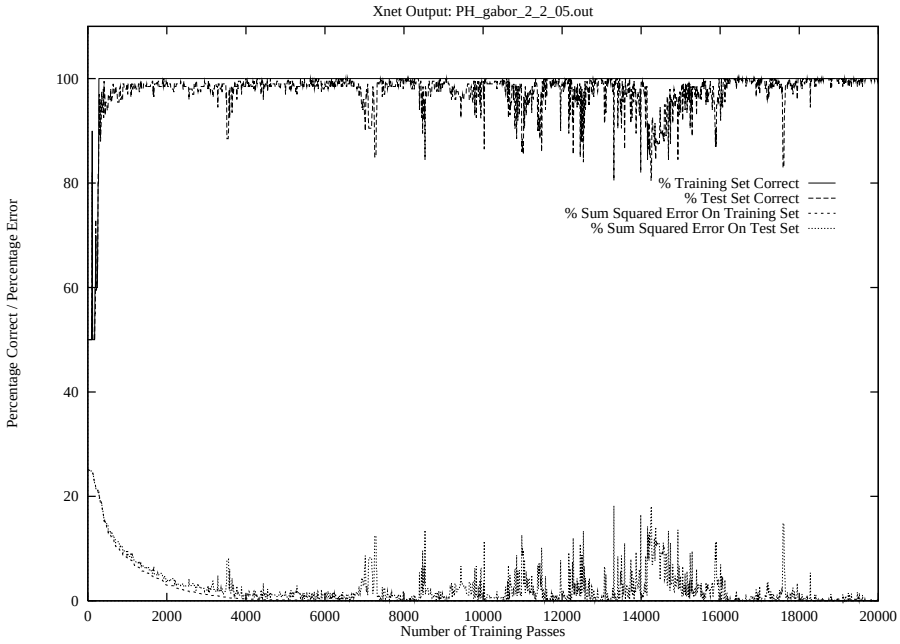
The use of simulated annealing for training means that the non-monotonic derivatives of the Gabor functions do not prevent the filter orientations from changing significantly during training. It should also be noted that using simulated annealing results in very much faster training, since no derivatives need to be evaluated. This is particularly significant in a case such as this where the derivatives of the Gabor layers involve trigonometric functions, which are expensive to compute.

Figure 4.6 shows the typical variation in training and test set errors and classification rates during training with simulated annealing. It can be seen that the behaviour can be very different depending upon the initial parameters of the network, and the stochastic nature of the algorithm. Table 4.7 shows the final percent errors for the networks.[3] Comparison with Table 4.6 indicates that in all cases where the final error on the training set was below 10%, 100% classification of the test set was achieved at some stage during training. Since simulated annealing will find the minimum on the training data given a sufficiently long annealing schedule and a sufficiently high initial temperature, it seems that a 100% correct solution should always be able to be found for such networks.

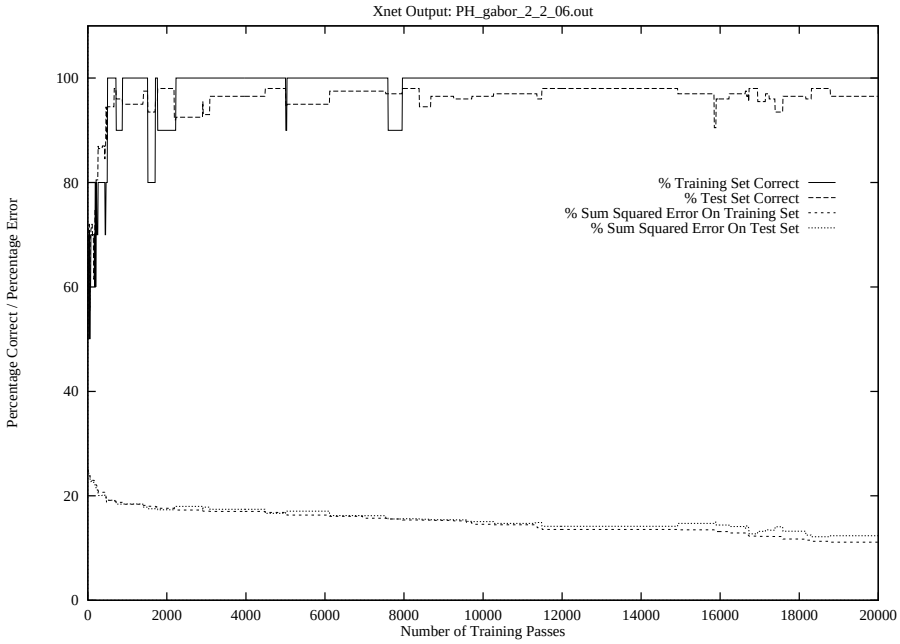
	Training Data	Test Data
Network 1	11.05476	16.26678
Network 2	0.00015	0.49714
Network 3	10.39662	15.41420
Network 4	0.00006	1.27624
Network 5	0.00079	0.06715
Network 6	11.13610	12.36650
Network 7	0.00024	0.30148
Network 8	0.00016	0.00016
Network 9	0.00058	4.00381
Network 10	0.89500	1.37818

Table 4.7: Final percent errors of the 22 parameter MBNN classifiers after 20000 iterations of simulated annealing training on the Riser/Non-riser problem.

[3] *i.e.* as a percentage of the maximum possible error, corresponding to "exactly wrong" classification of all patterns.



(a) Initial Gabor Orientations Good



(b) Initial Gabor Orientations Poor

Figure 4.6: Variation in training and test set errors and classification performance during training of a 22 parameter MBNN with simulated annealing.

4.5.2 Modified Backpropagation

A further 10 networks of this architecture, with the same initializations as those described above, were trained using the modified backpropagation algorithm. The first point to note, before considering the results in detail, is that this training was at least one (quite possibly two) orders of magnitude slower than that using simulated annealing. This may be attributed several causes. Most obviously, there is the computational cost due to the need to calculate derivatives containing trigonometric functions. The convergence of the algorithm was also often very slow, which may also be attributed to the sinusoidal terms in the derivatives. The error surface is far from monotonic, and a gradient descent algorithm such as backpropagation is not appropriate.

	Best		Final (25000 iterations)	
	Training Data	Test Data	Training Data	Test Data
Network 1	100.00	99.50	100.00	93.50
Network 2	100.00	99.00	100.00	99.00
Network 3	100.00	90.00	100.00	89.50
Network 4	90.00	80.50	90.00	80.00
Network 5	100.00	96.00	100.00	96.00
Network 6	100.00	97.50	100.00	93.50
Network 7	90.00	82.50	90.00	76.50
Network 8	100.00	93.00	100.00	93.00
Network 9	100.00	93.50	100.00	85.00
Network 10	90.00	81.00	90.00	79.50
$\mu \pm \sigma$	97.00± 4.48	91.25 ± 7.05	97.00± 4.58	88.55 ± 7.40

Table 4.8: Classification performance (% correct) of the 22 parameter MBNN classifiers after 25000 iterations of backpropagation training on the Riser/Non-riser problem.

The results appear in Table 4.8. As would be expected, the training set is learnt perfectly in nearly every case. The exceptions, such as Network 7, experienced a catastrophic failure during training, as can be seen in Figure 4.7. This sort of failure was responsible for the poorly-performed MBNNs reported in §4.3, and it seems that this simple modification of backpropagation is, in general, inadequate for weighting functions such as the Gabor.

Even in those instances in which such catastrophic failure did not occur, in no case was the final test set performance 100%. This may be partially attributed to overfitting of the training data. Figure 4.8 shows that the final test set performance for Network 1 was 93.5% after 25000 iterations, whereas it had been 99.5% after 1840 iterations. This also indicates the degree of variation in the number of iterations required for convergence between differently initialized networks.

The statistics show that the overall performance was significantly worse for the networks training using the modified backpropagation algorithm than for those trained

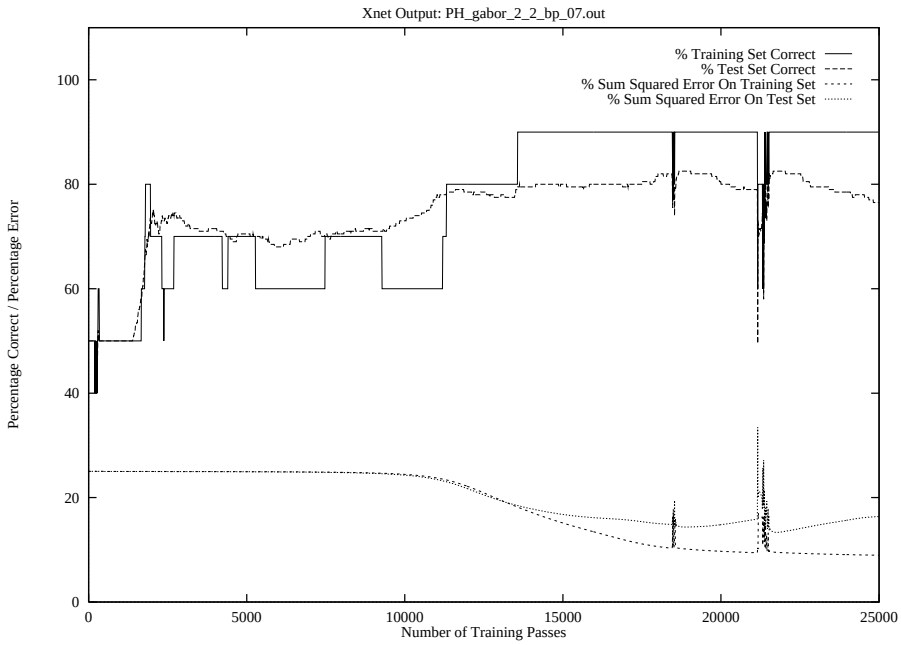


Figure 4.7: Catastrophic failure during training of a 22 parameter MBNN with backpropagation.

using simulated annealing. Given the non-monotonic structure imposed on the error surface this is unsurprising.

4.6 Chapter Summary

In this chapter we have introduced a particular form of MBNN: the MBNN in which weights are determined by parameterized weighting functions of the form described in §4.1. We have shown how the backpropagation algorithm can be naturally extended to apply to networks of this form, enabling the dimensionality of the parameter space searched during training to be much smaller than the number of connections in the network. Experimental results, however, indicate that a stochastic training algorithm, such as simulated annealing, is more appropriate in many cases.

The application of such networks was demonstrated on several tasks, indicating that they are indeed trainable, and that better invariance performance is achieved than with prior TNN techniques. Moreover, this is achieved at a greatly reduced computational cost, in terms of both the size of the required training set and the complexity of the search problem.

In the chapters ahead, a new shift-, rotation- and scale-invariant characteristic of two-dimensional contours is introduced, which is particularly suited to being embedded in a MBNN, since it involves inherently local calculations. It will be shown that another type of MBNN can be constructed that classifies on the basis of this characteristic, and

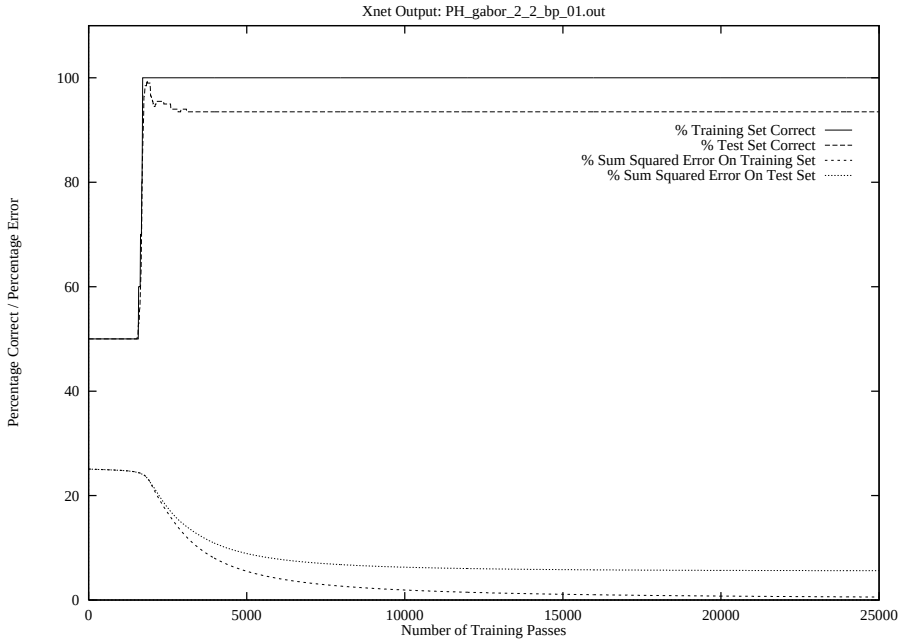


Figure 4.8: Overfitting of training data, resulting in diminished test set performance during training of a 22 parameter MBNN with backpropagation.

that such networks can be successfully applied to invariant optical character recognition.

Chapter 5

Invariance Signatures

The aim of any invariant pattern recognition technique is to obtain a representation of the pattern in a form that is invariant under some specified transformations of the original image. Ideally, such a technique would produce a representation of the pattern which was not only invariant, but which also uniquely characterized the input pattern.

This goal is not necessarily realisable, nor is it always as desirable as it might seem. In many pattern recognition problems, the aim is to produce a system which classifies input patterns as belonging to a particular *class*, rather than to identify uniquely every single input pattern presented. In such cases a unique representation for each and every possible input pattern can actually be a disadvantage. What is required is an invariant representation that retains enough information for distinct classes to be distinguished. It is by no means necessary that members of the same class be distinguishable in the invariant representation used.

Invariant pattern recognition has a long history, as was discussed in Chapter 2. One approach is to study the ways in which *local* features change under the action of *global* transformations of the image. Such considerations lead naturally to the study of Lie transformation groups, and many, varied invariant pattern recognition techniques make use of the Lie theory [Hoffman, 1966; Hoffman, 1978; Ferraro and Caelli, 1988; Cole, Murase and Naito, 1991; Rubinstein et al., 1991; Segman et al., 1992; Squire and Caelli, 1995].

In this chapter the theory of Lie transformation groups will be developed, and their significance for invariant pattern recognition made clear. A new shift-, rotation- and scale- invariant function of a two-dimensional contour with respect to a given Lie transformation group, the *Invariance Measure Density Function*, will then be derived. It will be shown that several such functions can be combined to yield an *Invariance Signature* for the contour. This Invariance Signature has several properties that make it attractive for implementation in a MBNN: it is based on local properties of the contour, so initial calculations are inherently parallel; it is statistical in nature, and its resolution

can be chosen at the designer's discretion, allowing direct control over the dimensionality of the network implementation. However, the use of the Invariance Signature is by no means limited to neural network implementations. Whilst the Invariance Signature is not "strongly-invariant" in the sense discussed in §2.1 (patterns are not uniquely represented), it will be shown in Chapter 7 that *classes* of patterns are represented sufficiently differently for optical character recognition applications.

5.1 Lie Transformation Groups and Invariance

5.1.1 Definition of a Group

A group is a set of elements and an associated operation which maps elements of the group into each other. Consider a set of elements G and an associated operation $\oplus$. Let the elements of G be denoted by $\epsilon_1, \epsilon_2, \ldots$. For G to be a group, it must have the following properties:

1. Closure:

$$\forall \ \epsilon_i, \epsilon_j \in G, \quad \epsilon_i \oplus \epsilon_j \in G \tag{5.1}$$

2. Associativity:

$$\forall \ i, j, k, \quad (\epsilon_i \oplus \epsilon_j) \oplus \epsilon_k = \epsilon_i \oplus (\epsilon_j \oplus \epsilon_k) \tag{5.2}$$

3. Identity Element I:

$$\exists \ I: \ \forall \ \epsilon \in G, \quad \epsilon \oplus I = \epsilon \tag{5.3}$$

4. Inverse Elements:

$$\forall \ \epsilon \in G, \ \exists \ \epsilon^{-1}: \quad \epsilon \oplus \epsilon^{-1} = I \tag{5.4}$$

5.1.2 One Parameter Lie Groups in Two Dimensions

An important class of groups for pattern recognition is the Lie transformation groups. A Lie group is a continuous transformation group with a differentiable structure. For two-dimensional invariant pattern recognition, the most interesting groups are the one-parameter Lie transformation groups defined on the plane. These transformation groups include rotation, dilation and translation: transformations with respect to which invariance is often desired in pattern recognition systems. These are smooth transformations

of the form

$$\begin{aligned} x' &= \mu(x, y, \alpha) \\ y' &= \nu(x, y, \alpha). \end{aligned} \tag{5.5}$$

The parameter α determines which element of the group the transformation is. For instance, if α_0 corresponds to the identity element, we have

$$\begin{aligned} x' &= \mu(x, y, \alpha_0) = x \\ y' &= \nu(x, y, \alpha_0) = y. \end{aligned} \tag{5.6}$$

There is a vector field $\vec{g} = \begin{bmatrix} g_x & g_y \end{bmatrix}^{\mathrm{T}}$ associated with each Lie group G. This vector field gives the direction in which a point (x, y) is "dragged" by an infinitesimal transformation under the action of the group. It is given by

$$\begin{aligned} g_x(x, y) &= \left.\frac{\partial \mu}{\partial \alpha}\right|_{\alpha=\alpha_0} \\ g_y(x, y) &= \left.\frac{\partial \nu}{\partial \alpha}\right|_{\alpha=\alpha_0}. \end{aligned} \tag{5.7}$$

This vector field $\vec{g}$ allows an operator $\mathscr{L}_G$ to be defined,

$$\mathscr{L}_G = g_x \frac{\partial}{\partial x} + g_y \frac{\partial}{\partial y}. \tag{5.8}$$

The operator $\mathscr{L}_G$ is called the *generator* of the transformation group G. It is called the generator of the group because it can be used to construct the finite transformation corresponding to the infinitesimal dragging described in Equations 5.7. This is done by "summing" the repeated applications of the infinitesimal transformation.

5.1.3 From Infinitesimal to Finite Transformations

Let us consider the case in which we know the direction of the vector field specifying the infinitesimal transformation at each point, as given by Equations 5.7, and we wish to construct the Equations 5.5 specifying the finite transformation. We will consider the transformation of x in detail. For a small change in the group parameter from the identity element, $\alpha = \alpha_0 + \Delta\alpha$, we can approximate the change in x by

$$x' = x + \Delta x \approx x + \Delta\alpha \left.\frac{\partial \mu}{\partial \alpha}\right|_{\alpha=\alpha_0} \tag{5.9}$$

We now wish to find a finite transformation corresponding to n applications of the $\Delta\alpha$ transformation. This will approximate the finite transformation corresponding to

the group element specified by parameter $\alpha = n\Delta\alpha$. Let x_i correspond to the value of x' after i applications of the transformation specified by $\Delta\alpha$. We obtain

$$\begin{aligned} x_0 &= x \\ x_1 &= x_0 + \frac{\alpha}{n}\mathscr{L}_G x_0 = \left(1 + \frac{\alpha}{n}\mathscr{L}_G\right) x \\ x_2 &= x_1 + \frac{\alpha}{n}\mathscr{L}_G x_1 \\ &= \left(1 + \frac{\alpha}{n}\mathscr{L}_G\right) x_1 \\ &= \left(1 + \frac{\alpha}{n}\mathscr{L}_G\right)^2 x \end{aligned} \tag{5.10}$$

and thus

$$x_n = \left(1 + \frac{\alpha}{n}\mathscr{L}_G\right)^n x.$$

In the limit as $n \to \infty$, the approximation becomes exact, and the finite transformation is given by

$$\mu(x, y, \alpha) = \lim_{n\to\infty} \left(1 + \frac{\alpha}{n}\mathscr{L}_G\right)^n x. \tag{5.11}$$

Similarly,

$$\nu(x, y, \alpha) = \lim_{n\to\infty} \left(1 + \frac{\alpha}{n}\mathscr{L}_G\right)^n y. \tag{5.12}$$

5.1.4 Derivation of the Rotation Transformation

As an example, we will derive the rotation transformation from the knowledge only that each point P should be dragged in a direction at right angles to the line from the origin to P by the infinitesimal transformation, as shown in Figure 5.1. We will denote the rotation group R. The parameter of the group is θ. From Figure 5.1, we see that

$$\begin{aligned} dx &= -r d\theta \sin\phi \\ &= -\sqrt{x^2 + y^2} d\theta \frac{y}{\sqrt{x^2 + y^2}} \\ &= -y d\theta. \end{aligned} \tag{5.13}$$

Similarly,

$$dy = x d\theta. \tag{5.14}$$

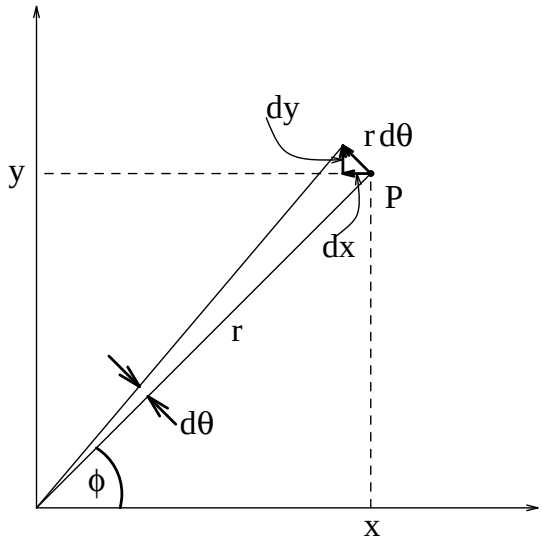


Figure 5.1: Infinitesimal transformation leading to rotation.

The generator of the rotation group R is thus

$$\mathscr{L}_R = -y\frac{\partial}{\partial x} + x\frac{\partial}{\partial y}. \tag{5.15}$$

Consequently, the corresponding finite transformation for x is

$$\mu(x, y, \theta) = \lim_{n\to\infty}\left(1 + \frac{\theta}{n}\mathscr{L}_R\right)^n x. \tag{5.16}$$

Using the binomial expansion,

$$\mu(x, y, \theta) = \lim_{n\to\infty}\left[1 + n\frac{\theta}{n}\mathscr{L}_R + \frac{n(n-1)}{2!}\frac{\theta^2}{n^2}\mathscr{L}_R^2 + \frac{n(n-1)(n-2)}{3!}\frac{\theta^3}{n^3}\mathscr{L}_R^3 + \frac{n(n-1)(n-2)(n-3)}{4!}\frac{\theta^4}{n^4}\mathscr{L}_R^4 + \cdots\right]x. \tag{5.17}$$

We note that

$$\lim_{n\to\infty}\frac{n!}{(n-k)!n^k} = 1, \tag{5.18}$$

and

$$\begin{aligned}\mathscr{L}_R x &= -y\\ \mathscr{L}_R^2 x &= -x\\ \mathscr{L}_R^3 x &= y\\ \mathscr{L}_R^4 x &= x,\end{aligned} \tag{5.19}$$

after which the values repeat. We can therefore write

$$\begin{aligned}\mu(x,y,\theta) &= \left[x - \theta y - \frac{\theta^2}{2!}x + \frac{\theta^3}{3!}y + \frac{\theta^4}{4!}x + \cdots\right] \\ &= x\left(1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \cdots\right) - y\left(\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \cdots\right) \\ &= x\cos\theta - y\sin\theta.\end{aligned} \tag{5.20}$$

Similarly,

$$\nu(x,y,\theta) = x\sin\theta + y\cos\theta. \tag{5.21}$$

These are the well-known equations describing a rotational transformation by an angle θ. The procedure above can be used to generate the finite transformation corresponding to any Lie operator.

5.1.5 Functions Invariant Under Lie Transformations

A function is said to be invariant under the action of a transformation if all points of the function are mapped into other points of the function by the action of the transformation on the coordinate system. Such functions are important in pattern recognition, since it is often desirable to recognize patterns independently of common image transformations such as rotation, dilation and scaling.

Consider a function $F(x,y)$. We wish to determine its invariance with respect to a Lie transformation group G. Let

$$\vec{g}(x,y) = \begin{bmatrix} g_x(x,y) & g_y(x,y) \end{bmatrix}^{\mathrm{T}} \tag{5.22}$$

be the vector field corresponding to the generator of the Lie group, $\mathscr{L}_G$. F is constant with respect to the action of the generator if

$$\mathscr{L}_G F = 0. \tag{5.23}$$

This can be written in terms of the vector field as

$$\nabla F \cdot \vec{g}(x,y) = 0, \tag{5.24}$$

that is

$$\frac{\partial F}{\partial x}g_x + \frac{\partial F}{\partial y}g_y = 0, \tag{5.25}$$

or

$$\frac{\frac{\partial F}{\partial x}}{\frac{\partial F}{\partial y}} = -\frac{g_y}{g_x}. \tag{5.26}$$

Now consider a contour C parameterized by t on which F is constant,

$$\forall t \;\; F(x(t), y(t)) = K. \tag{5.27}$$

Since F is constant on the contour, we have

$$\frac{dF}{dt} = \frac{\partial F}{\partial x}\frac{dx}{dt} + \frac{\partial F}{\partial y}\frac{dy}{dt} = 0, \tag{5.28}$$

and consequently

$$\frac{\frac{\partial F}{\partial x}}{\frac{\partial F}{\partial y}} = -\frac{dy}{dx}. \tag{5.29}$$

Thus the condition for invariance with respect to the Lie transformation group generated by $\mathscr{L}_G$, given in Equation 5.24, holds if

$$\frac{dy}{dx} = \frac{g_y}{g_x}. \tag{5.30}$$

on the contour.

The condition derived in Equation 5.30 has a very natural interpretation. It says that a contour is invariant under the action of a group G if the tangent to the contour at each point is in the same direction as the vector field $\vec{g}$ corresponding to the infinitesimal transformation that generates the group.

5.2 From Local Invariance Measures to Global Invariance

We now propose a new shift-, rotation- and dilation-invariant signature for contours. We call this an *Invariance Signature*, since it is derived from the degree to which a given contour is consistent with invariance under a set of Lie transformation groups.

5.2.1 The Local Measure of Consistency

We have seen in Equation 5.30 that in order for a contour C to be invariant under a transformation group G the tangent to the contour must be everywhere parallel to the vector field defined by the generator of the group. We now define the *Local Measure of Consistency* with invariance under a transformation group G at a point (x, y) on C,

$\iota_G(x, y)$.

$$\iota_G(x, y) = \left|\hat{\theta}(x, y) \cdot \hat{g}_G(x, y)\right| \tag{5.31}$$

The absolute value is used because only the orientation of the tangent vector is significant, not the direction. At each point both the tangent vector to the contour, $\vec{\theta}(x, y)$ and the vector field $\vec{g_G}(x, y)$ are normalized:

$$\hat{g}_G(x, y) = \frac{g_x(x, y)\hat{\imath} + g_y(x, y)\hat{\jmath}}{\sqrt[+]{g_x^2(x, y) + g_y^2(x, y)}} \tag{5.32}$$

and

$$\hat{\theta}(x, y) = \frac{\hat{\imath} + \frac{dy}{dx}\hat{\jmath}}{\sqrt[+]{1 + \left(\frac{dy}{dx}\right)^2}}, \tag{5.33}$$

where $\hat{\imath}$ and $\hat{\jmath}$ are unit vectors in the x and y directions respectively. Substituting Equations 5.32 and 5.33 in Equation 5.31, we obtain

$$\iota_G(x, y) = \frac{1}{\sqrt[+]{1 + \left[\frac{g_y(x,y) - g_x(x,y)\frac{dy}{dx}}{g_x(x,y) + g_y(x,y)\frac{dy}{dx}}\right]^2}} \tag{5.34}$$

5.2.2 The Invariance Measure Density Function

Equation 5.31 is a mapping $C \mapsto [0, 1]$, which gives a measure of the degree to which the tangent at each point is consistent with invariance under G. We wish to find a function which characterizes the degree to which the entire contour C is consistent with invariance under G. Such a function is the density function for the value of ι_G in $[0, 1]$, $I(\iota_G)$, which we will call the *Invariance Measure Density Function*. The more points from C that are mapped to values close to 1 by Equation 5.31, the more consistent C is with invariance under G. The shape of $I(\iota_G)$ is a descriptor of C, and we will show that $I(\iota_G)$ is invariant under rotations and dilations of C (see Theorem 5.1). Translation invariance is obtained if the origin of coordinates in which ι_G is calculated is chosen to be the centroid of C.

It is interesting to note that there is evidence from psychophysical experiments that a measure of the *degree of invariance* of a pattern with respect to the similarity group of transformations (rotations, translations and dilations) is important in human pattern recognition [Caelli and Dodwell, 1984]. The measure proposed here might be seen as a mathematical formalization of this notion. Moreover, its implementation in a neural network architecture is consistent with Caelli and Dodwell's (1984, p. 159) statement

of a proposal due to Hoffman [1966; 1978]:

> Hoffman's fundamental postulate was that the coding of orientation at various positions of the retinotopic map by the visual system, discovered by Hubel and Wiesel (1962) and others, actually provides the visual system with "vector field" information. That is, the visual system, on detecting specific orientation and position states ("Φ/P codes"), spontaneously extracts the path curves (interpreted as visual contours) of which the local vectors are tangential elements.

First, however, we must establish the form of $I(\iota_G)$. Without loss of generality, consider C to be parameterized by $t : t \in [t_0, T]$. The arc length s along C to a given value of the parameter t is:

$$s(t) = \int_{t_0}^{t} \sqrt{\left(\frac{dx}{dt}\bigg|_\tau\right)^2 + \left(\frac{dy}{dt}\bigg|_\tau\right)^2}\, d\tau. \tag{5.35}$$

The total length of C is thus $S = s(T) = \oint_C ds$. For well-behaved functions $F(x, y)$, we can construct $s(t)$ such that we can reparameterize C in terms of s. Thus we can rewrite Equation 5.31 to give ι_G in terms of s. For simplicity, we will first consider the case in which ι_G is a monotonic function of s, as shown in Figure 5.2.

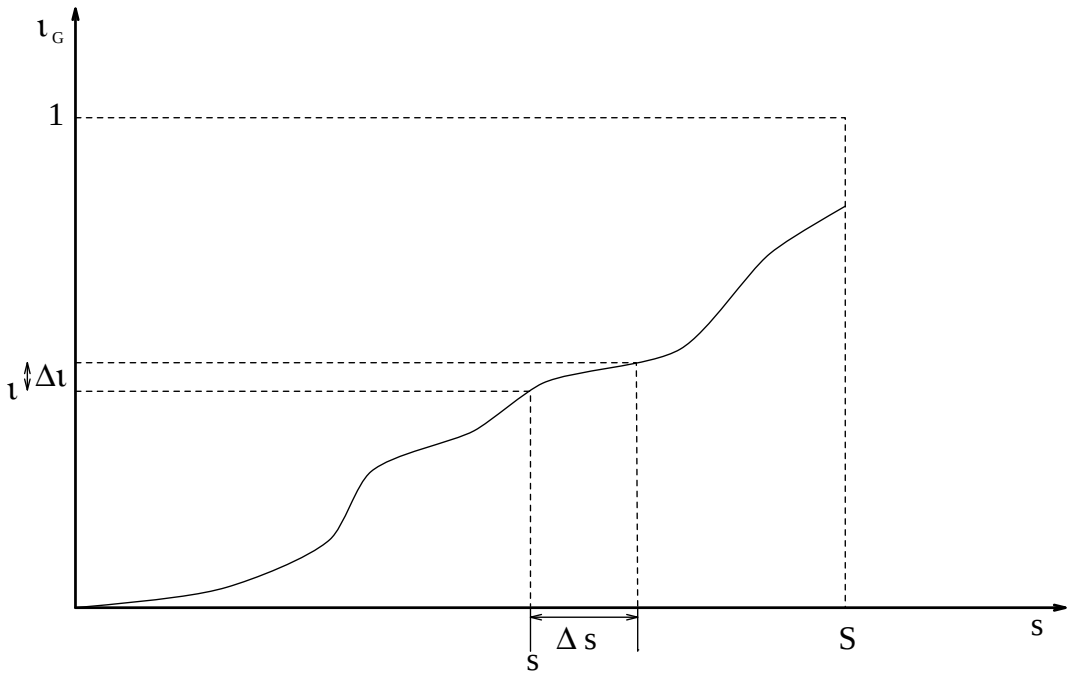


Figure 5.2: Local Measure of Consistency as a function of arc length.

The Invariance Measure Density is:

$$\begin{aligned} I(\iota_G) &= \lim_{\Delta\iota_G \to 0} \left| \frac{\Delta s}{S \Delta \iota_G} \right| \\ &= \frac{1}{S} \left| \frac{ds}{d\iota_G} \right|. \end{aligned} \tag{5.36}$$

$I(\iota_G)$ can be interpreted as the probability density function for ι_G at points $(x(s), y(s))$, where s is a random variable uniformly distributed on $[0, S]$. It is clear that for the general case, in which ι_G is not a monotonic function of s, the function could be broken up into piecewise monotonic intervals, and the contribution to the density from each interval summed. The general form for a specific value $\iota_G{}'$ is therefore

$$I\left(\iota_G{}'\right) = \frac{1}{S} \sum_{s \in [0,S]: \iota_G(s) = \iota_G{}'} \left| \frac{ds}{d\iota_G} \right|. \tag{5.37}$$

Theorem 5.1 *The Invariance Measure Density, $I(\iota_G)$, is invariant under translations, rotations and dilations of the contour C with respect to which it is calculated.*

Proof That $I(\iota_G)$, defined in Equation 5.37, is invariant under translations of the contour C is trivial, since, as defined in §5.2.2, ι_G is calculated with the origin at the centroid of the contour C. Rotation and dilation invariance can be proved by construction. Since the transformations for rotation and dilation in the plane commute,[1] we can consider a two-parameter Abelian group corresponding to a rotation by an angle θ and a dilation by a positive factor β. The coordinates x and y are transformed according to

$$\begin{aligned} x' &= \beta(x \cos\theta - y \sin\theta) \\ y' &= \beta(x \sin\theta + y \cos\theta). \end{aligned} \tag{5.38}$$

We are interested in the relationship between the arc length function $s(t)$ defined in Equation 5.35 for the original parameterized contour $(x(t), y(t))$ and the new function $s'(t)$ when the coordinates are transformed according to Equation 5.38. We find that

$$\begin{aligned} \frac{dx'}{dt} &= \beta \cos\theta \frac{dx}{dt} - \beta \sin\theta \frac{dy}{dt} \\ \frac{dy'}{dt} &= \beta \sin\theta \frac{dx}{dt} + \beta \cos\theta \frac{dy}{dt}. \end{aligned} \tag{5.39}$$

[1] As a consequence of Clairaut's theorem, they commute when the functions to which the transformations are applied are both defined and continuous in a neighbourhood around each point of the function.

Combining these, we find that

$$\begin{aligned}\left(\frac{dx'}{dt}\right)^2+\left(\frac{dy'}{dt}\right)^2 =&\beta^2\left[\cos^2\theta\left(\frac{dx}{dt}\right)^2-2\cos\theta\sin\theta\frac{dx}{dt}\frac{dy}{dt}+\sin^2\theta\left(\frac{dy}{dt}\right)^2\right.\\ &\left.+\sin^2\theta\left(\frac{dx}{dt}\right)^2+2\cos\theta\sin\theta\frac{dx}{dt}\frac{dy}{dt}+\cos^2\theta\left(\frac{dy}{dt}\right)^2\right]\\ =&\beta^2\left[\left(\frac{dx}{dt}\right)^2+\left(\frac{dy}{dt}\right)^2\right]. \end{aligned} \tag{5.40}$$

This result can be substituted into Equation 5.35 to obtain

$$s'(t)=\beta s(t). \tag{5.41}$$

This clearly indicates that the total arc length is $S'=\beta S$, and the derivative of $s(t)$ is also scaled. Substituting into Equation 5.37, we obtain

$$\begin{aligned} I'\left(\iota_G{}'\right) &= \frac{1}{S'}\sum_{\iota_G(s)=\iota_G{}'}\left|\frac{ds'}{d\iota_G}\right| \\ &= \frac{1}{\beta S}\sum_{\iota_G(s)=\iota_G{}'}\beta\left|\frac{ds}{d\iota_G}\right| \\ &= \frac{1}{S}\sum_{\iota_G(s)=\iota_G{}'}\left|\frac{ds}{d\iota_G}\right| \\ &= I\left(\iota_G{}'\right) \\ &\quad Q.E.D. \end{aligned} \tag{5.42}$$

Thus we have demonstrated that $I\left(\iota_G{}'\right)$ is invariant under rotations and dilations of the contour C.

Invariance Measure Densities For Specific Contours

In order to demonstrate how the Invariance Measure Density defined above can be applied, we will evaluate $I\left(\iota_G\right)$ for a specific contour.

The Square Let the contour C be a square of side $2L$ centred at the origin, as shown in Figure 5.3. We will find the Invariance Measure Density for C with respect to rotation, $I_{C_{rot}}(\iota)$. By symmetry, we need only find $I_{C_{rot}}$ for one side of the square. On the side indicated by the dashed line in Figure 5.3, x and y can be expressed in

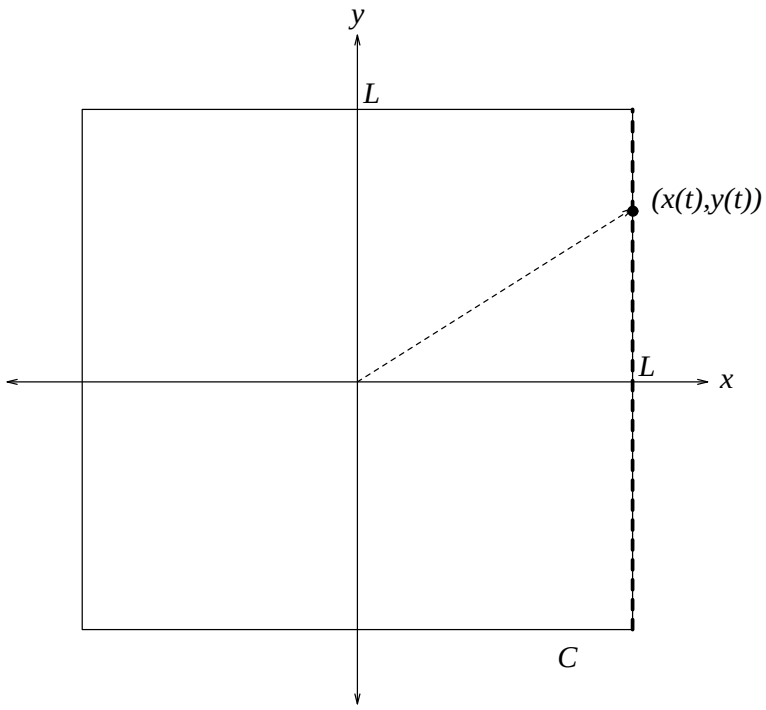


Figure 5.3: Square of side 2L.

terms of a parameter t as

$$\begin{aligned} x &= L \\ y &= t, \qquad -L \leq t \leq L. \end{aligned} \tag{5.43}$$

Here the arc length $s(t) = t + L$, and the total is $S = 2L$. If $\dot{x}(t)$ and $\dot{y}(t)$ are the derivatives of x and y with respect to t, Equation 5.34 can be rewritten as

$$\iota_{C_{rot}}(s) = \frac{1}{\sqrt{1 + \left[\frac{g_y(s)\dot{x}(s) - g_x(s)\dot{y}(s)}{g_x(s)\dot{x}(s) + g_y(s)\dot{y}(s)}\right]^2}} \tag{5.44}$$

Here we have $\dot{x}(t) = 0$ and $\dot{y}(t) = 1$. Equations 5.15 and 5.43 can be substituted into Equation 5.44 to give

$$\iota_{C_{rot}}(s) = \frac{1}{\sqrt{1 + \left(\frac{s-L}{L}\right)^2}}, \tag{5.45}$$

which can be inverted to yield

$$s = L\left(1 + \sqrt{\frac{1}{\iota^2_{C_{rot}}} - 1}\right); \tag{5.46}$$

differentiating,

$$\frac{ds}{d\iota_{C_{rot}}} = \frac{-L}{\iota_{C_{rot}}^2\left(1-\iota_{C_{rot}}^2\right)}. \tag{5.47}$$

Using Equation 5.37, we arrive at our final result:

$$\begin{aligned} I_{C_{rot}}\left(\iota_{C_{rot}}\right) &= \frac{1}{2L}\sum_{\iota'_{C_{rot}}=\iota_{C_{rot}}}\left|\frac{ds}{d\iota_{C_{rot}}}\right|_{\iota'_{C_{rot}}} \\ &= \frac{1}{2L}\times 2\times\frac{L}{\iota_{C_{rot}}^2\left(1-\iota_{C_{rot}}^2\right)} \\ &= \frac{1}{\iota_{C_{rot}}^2\left(1-\iota_{C_{rot}}^2\right)}, \qquad \iota_{C_{rot}}\in\left[\frac{1}{\sqrt{2}},1\right]. \end{aligned} \tag{5.48}$$

Note that, as required, the scale of the square L does not appear in this result. The factor of 2 arises because $\iota_{C_{rot}}$ is a symmetric function of s, so the sum has two terms. This function, shown in Figure 5.4, is characteristic of the square.

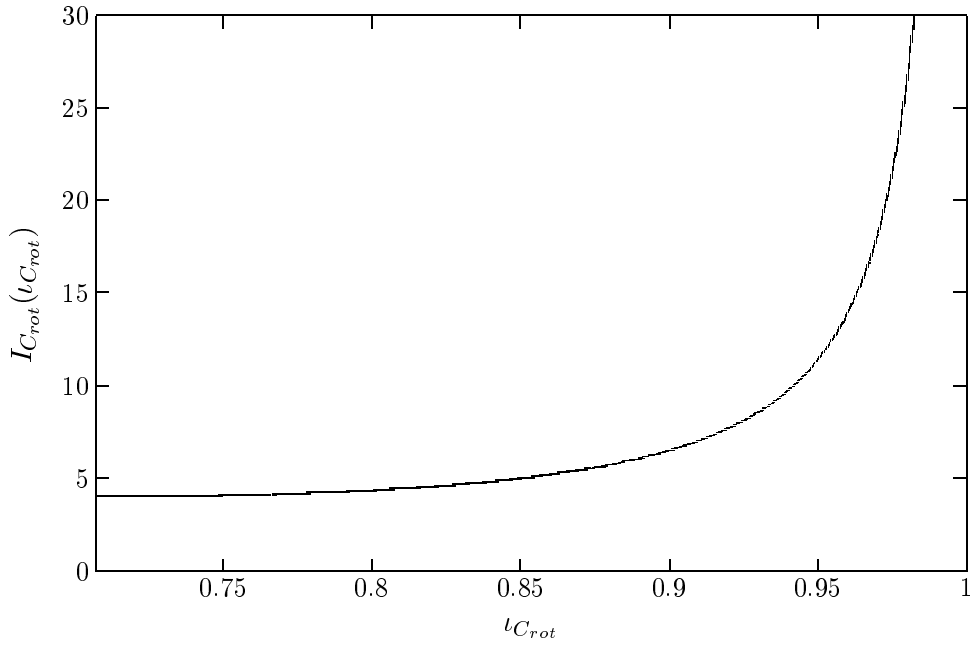


Figure 5.4: Invariance Density Measure with respect to rotation for a square.

5.2.3 Invariance Space: Combining Invariance Measure Densities

We have seen how to compute the Invariance Measure Density Function for a contour C with respect to invariance under a Lie transformation group G. We now consider the case in which the Invariance Measure Density Function is calculated with respect to a number of groups, and the results combined to provide a more complete characterization

of the transformational properties of a contour C. This operation can be considered to be a mapping of each point from the two dimensional image space to the interior of a unit hypercube in an n-dimensional *invariance space*, where each of the n dimensions corresponds to a particular sort of invariance. Equation 5.49 shows this for the case of a three-dimensional invariance space, where the dimensions correspond to the Local Measure of Consistency ι with respect to the transformations of rotation, dilation and translation:

$$(x, y) \mapsto \left[\begin{array}{ccc} \iota_{rot} & \iota_{dil} & \iota_{trans} \end{array} \right]^{\mathrm{T}}. \tag{5.49}$$

The distribution of points in this invariance space is characteristic of the contour C. This particular three-dimensional invariance space is the one that will be used for the experimental application of Invariance Signatures. Since each of the component invariance measure densities is invariant, this n-dimensional Invariance Signature is invariant under rotations, dilations, translations and reflections of the input image.

Vector Fields Corresponding to Rotation, Dilation and Translation Groups

The vector fields for the generators of the transformation groups for rotation, dilation and translation are given in normalized form. All can be derived using Equation 5.7: for rotation invariance

$$\vec{g}_{rot}(x, y) = \frac{1}{\sqrt{x^2 + y^2}} \left[\begin{array}{cc} -y & x \end{array} \right]^{\mathrm{T}}, \tag{5.50}$$

and for dilation invariance

$$\vec{g}_{dil}(x, y) = \frac{1}{\sqrt{x^2 + y^2}} \left[\begin{array}{cc} x & y \end{array} \right]^{\mathrm{T}}. \tag{5.51}$$

The translation invariance case is somewhat different. In fact, the term "translation invariance" is something of a misnomer. What is in fact measured in this case is the degree to which the contour is "linear". The vector field used is constant for all (x, y), and the direction is determined by finding the eigenvector corresponding to the dominant eigenvalue, $\vec{e}_1$, of the coordinate covariance matrix of all the points in the contour.[2] The direction of this eigenvector is the principal direction of the contour.

To place this in the same context as the previous two cases, the "translation invariance" factor is a measure of the degree to which the tangents at each point in the contour are consistent with a function invariant under translation in the direction of the vector $\vec{e}_1$. Since $\vec{e}_1$ is calculated from the image each time it is required, this measure is invariant under rotations, dilations and translations of the image. The vector field

[2] $\vec{e}_1$ is chosen to be a unit vector.

for the translation invariance case is thus:

$$\vec{g}_{trans}(x,y) = \left[\begin{array}{cc} e_{1x} & e_{1y} \end{array} \right]^{\mathrm{T}} \tag{5.52}$$

Uniqueness

It should be noted that this representation of the image is not unique, unlike some previous integral transform representations [Ferraro and Caelli, 1994]. The combination of individual Invariance Measure Densities into an Invariance Space does, however, increase the discriminant properties. As an example, removing two opposite sides of a square will not alter its rotation and dilation Invariance Signatures, but it will change the translation Invariance Signature. Likewise, a single straight line has the same translation Invariance Signature as any number of parallel straight lines, however they are spaced. The rotation and dilation Invariance Signatures, however, are sensitive to these changes.

5.2.4 Discrete Invariance Signatures

For any computer application of Invariance Signatures, a discrete version must be developed. The natural choice is the frequency histogram of the value of ι_G. For a continuous contour, this is obtained by dividing the interval $[0,1]$ into n "bins" and integrating $I(\iota_G)$ over each bin. For bins numbered from b_0 to b_{n-1}, the value in bin k is thus

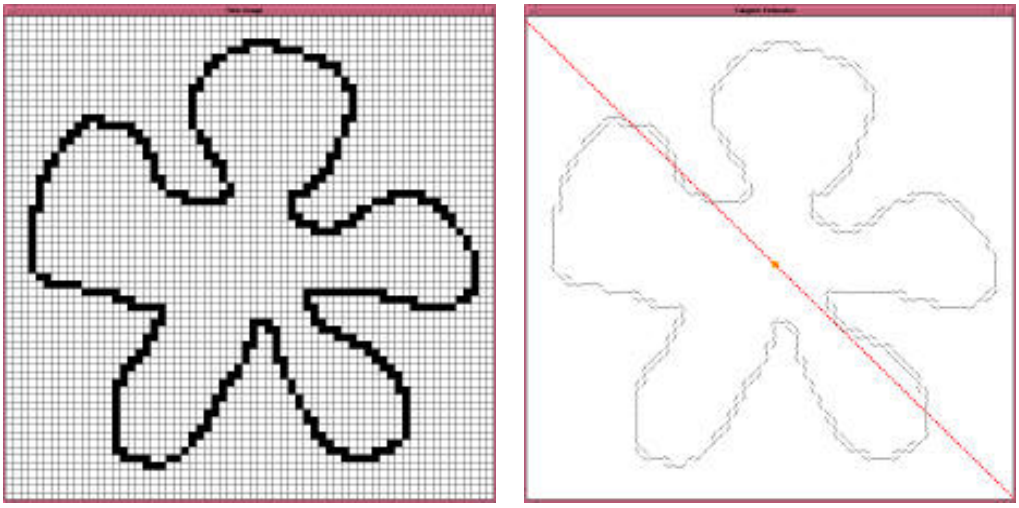
$$b_k = \int_{\frac{k}{n}}^{\frac{k+1}{n}} I(\iota_G) d\iota_G. \tag{5.53}$$

Since $I(\iota_G)$ is a probability density function, the sum of the values of the bins must be one.

For a system using images of sampled contours, a true frequency histogram of the estimated local measures of consistency may be used. The designer of a system using these Invariance Signatures must choose the number of bins n into which the data is grouped. It will be seen in Chapter 7 that this choice is not arbitrary for real systems.

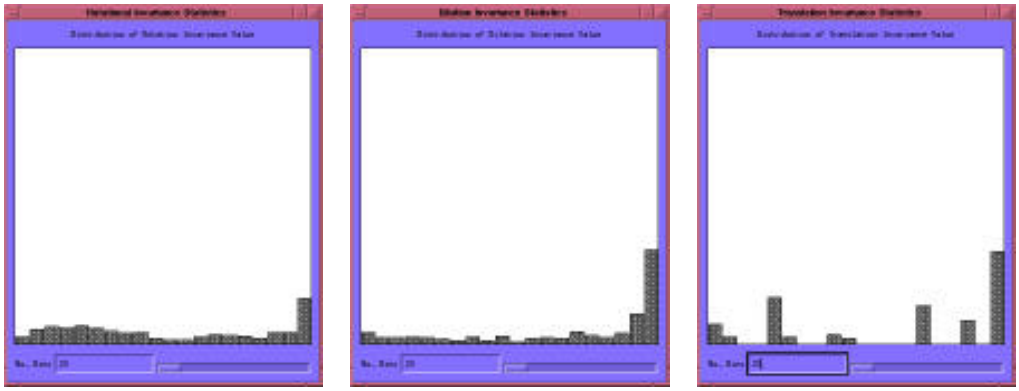
An example of a sampled contour and the estimated tangent vectors at each point is shown in Figure 5.5. The circle indicates the centroid of the contour, and the dashed line shows the direction of $\vec{e}_1$. The estimated discrete Invariance Signatures are shown in Figure 5.6, for 20 bins.

It would be expected that this "flower"-shaped contour would have Invariance Signatures which reflect a quite strong dilation-invariant component corresponding to the approximately radial edges of the "petals", and also a significant rotation-invariant component due to the ends of the petals which are approximately tangential to the



1. Original Contour 2. Tangent Estimates

Figure 5.5: Example of a sampled contour and its estimated tangents.



1. Rotation 2. Dilation 3. Translation

Figure 5.6: 20 bin discrete Invariance Signatures for the contour in Figure 5.5.

radial edges. This is indeed what is observed in Figure 5.6.

Chapter 6

A Neural Network for Computing Invariance Signatures

6.1 The Invariance Signature Neural Network Classifier

We propose a Model-Based Neural Network to compute the discrete Invariance Signature of an input pattern, as described in §5.2.4, and to classify the pattern on that basis. This MBNN consists of a fairly complex system of neural network modules, some hand-coded and some trained on sub-tasks. A schematic diagram is shown in Figure 6.1. This system, first described in Squire and Caelli (1995), will be referred to as the Invariance Signature Neural Network Classifier (ISNNC).

Whilst the ISNNC appears complex, it retains the basic characteristics of a traditional feed-forward neural network. It consists entirely of simple nodes joined by weighted connections.[1] Each node i in the network computes the sum of its j weighted inputs, net_i,

$$net_i = \sum_j w_{ij} x_j. \tag{6.1}$$

This is then used as the input to the transfer function f of the node, which is either linear, $f(net_i) = net_i$, or the standard sigmoid,

$$f(net_i) = \frac{1}{1 + e^{-net_i}}. \tag{6.2}$$

The only departure from a traditional neural network at runtime is that some of the

[1] with the exception of the *Dominant Image Orientation Unit* for which a neural network solution is still to be developed.

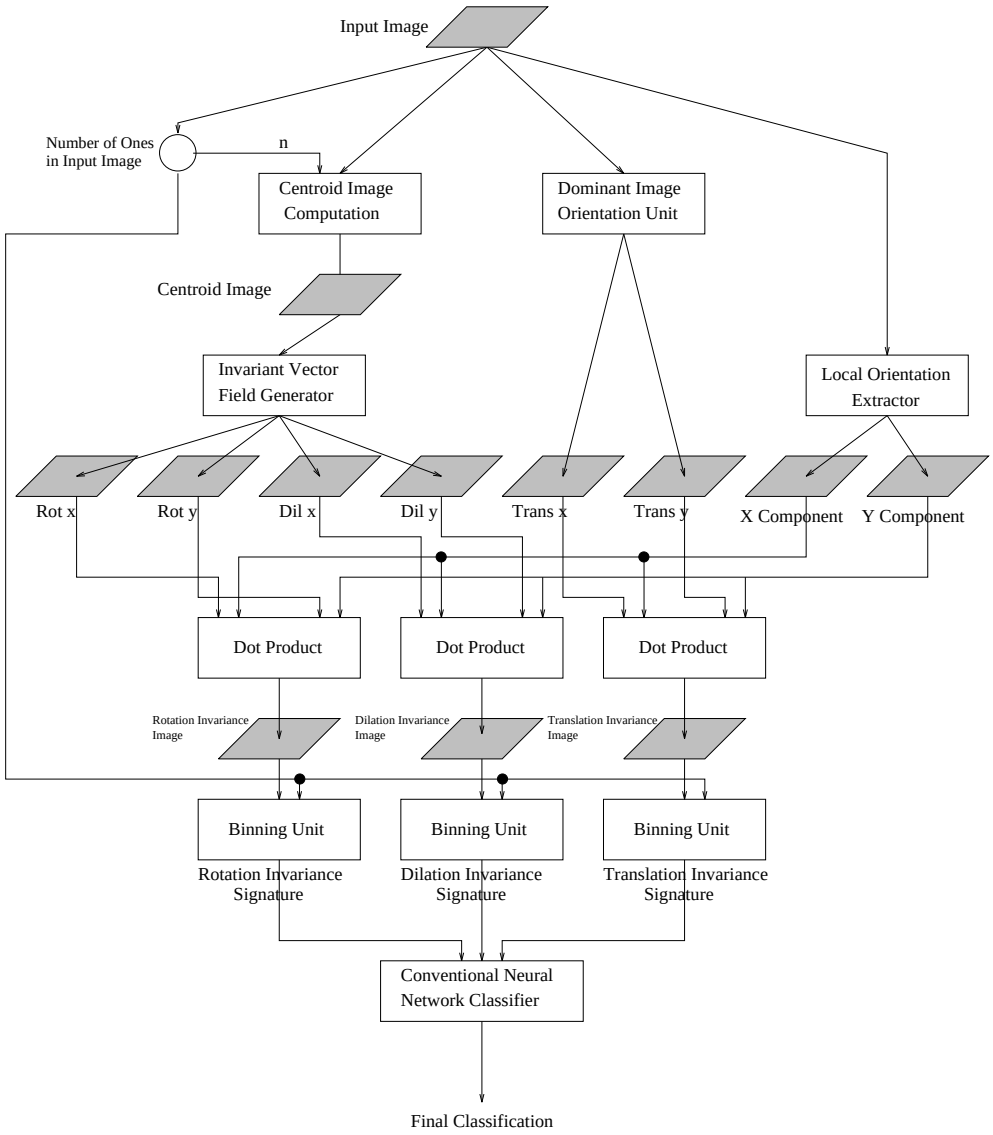


Figure 6.1: Invariance Signature-based contour recognition system.

connection weights are calculated by a node higher up in the network. We call these *dynamic weights*. The presence of dynamic weights allows the ISNNC to compute dot products,[2] and also for some nodes to act as gates controlling whether or not the output of a node is transmitted. Since connection weights in any implementation of a neural network are no more than references to some stored value, this should not present any difficulty.[3] Alternatively, the same functionality can be achieved by allowing a new class of node which multiplies (possibly weighted) inputs. Nodes of this type are

[2]The calculation of dot products, for example, is achieved by having the outputs of one layer provide the *weights* on connections to another layer.

[3]The aim of our research with MBNNs has never been biological plausibility. Perhaps this runtime weight calculation can be thought of as real-time, rapid, accurate training.

a component, for instance, of Higher Order Networks [Perantonis and Lisboa, 1992; Redding et al., 1993; Schmidt and Davis, 1993; Spirkovska and Reid, 1994; Delopoulos et al., 1994].

The ISNNC consists of a number of distinct levels. Data is processed in a feed-forward manner, and computation at a given level must be completed before computation at the next level can begin. Each of the following sections describes computation at a particular level.

6.1.1 Lie Vector Field Generation

Calculation of a Centroid Image

The first step in creating the vector fields corresponding to the generators of the rotation and dilation groups is to compute the coordinates of the centroid of the image, since, as seen in §5.2.2, this must be the origin of coordinates. This is done using a neural module which takes as its input a binary image, and outputs an image of the same size which is zero everywhere except at the centroid, where it is one. In Figure 6.1 this is labeled "Centroid Image". The weights of the Centroid Image Computation module are entirely hand-coded.

A quantity needed for this operation, and also later in the ISNNC, is N_{on}, the total number of "on" pixels in the input binary image $\mathcal{I}(x_i, y_j)$. This is given by

$$N_{\text{on}} = \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} \mathcal{I}(x_i, y_j). \tag{6.3}$$

This can easily be computed by a single node with a linear transfer function which has inputs from all input nodes with weights of 1. In fact, the value used in later levels of the ISNNC is $\frac{1}{N_{\text{on}}}$. Since the size of the input image is fixed, it is trivial to build a neural module which interpolates the function $f(x) = \frac{1}{x}$ with any desired accuracy for the integer values $\{x : x \in [0, N_x N_y]\}$. Bulsari (1993), for instance, demonstrates how to construct piecewise constant approximations to problems such as this with neural networks. From now on it will be assumed that the value $\frac{1}{N_{\text{on}}}$ is available.

The equations for the coordinates, $(\bar{x}, \bar{y})$, of the centroid of $\mathcal{I}(x_i, y_j)$ are

$$\begin{aligned} \bar{x} &= \frac{1}{N_{\text{on}}} \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} \mathcal{I}(x_i, y_j)\, x_i \\ \bar{y} &= \frac{1}{N_{\text{on}}} \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} \mathcal{I}(x_i, y_j)\, y_i \end{aligned} \tag{6.4}$$

A simple system of two linear neurons can compute each of these quantities. Such a

module is shown in Figure 6.2. This illustrates the notion of dynamic weights mentioned above. The weight on the single connection between Node A and Node B is dynamic, being computed from the number of ones in the input image. The weights on the connections to Node A are the x-coordinates of each of the input nodes.[4] These weights are fixed during the design process, so there is no sense in which a node needs to "know" what its coordinates are. An identical system is used to compute $\bar{y}$.

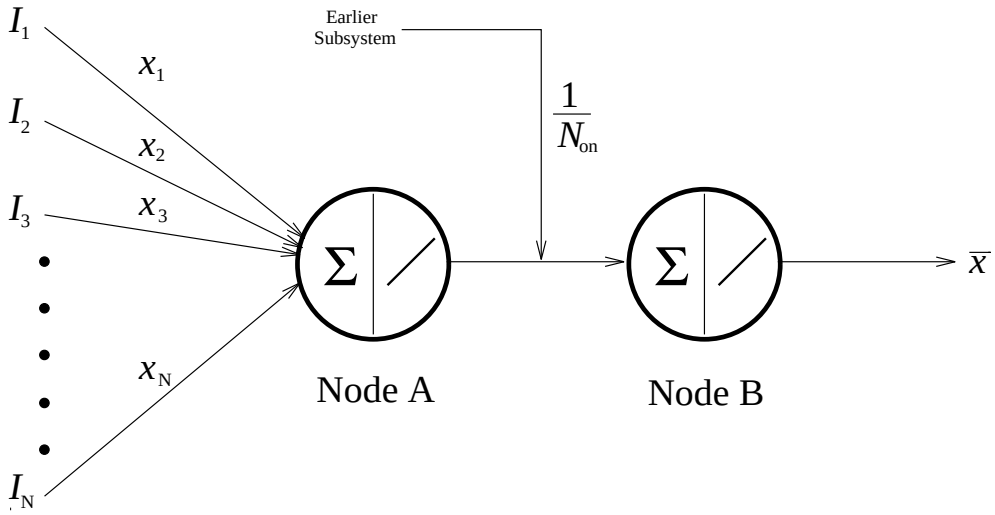


Figure 6.2: $\bar{x}$ module.

Once $\bar{x}$ and $\bar{y}$ have been calculated, the Centroid Image can be generated. This is done using a pair of nodes for each input node, as shown in Figure 6.3.[5] The weights and inputs to Nodes A and B are set so that they only both "fire" if their coordinate is within θ of the centroid coordinate. Here $\theta = 0.5$ was used, since the input nodes were assigned unit spacing. A neural AND of these nodes is then calculated by Node C.

This results in two images, one for the x coordinate and one for y. Each has a straight line of activation corresponding to the centroid value for that coordinate. A neural AND of these images is computed, giving the final output Centroid Image, $\mathcal{I}_C$, in which only the centroid node is one.

Gating of Vector Fields

Once the Centroid Image has been calculated, it is used to produce the rotation and dilation vector fields, four images in all. The weights from the Centroid Image are all derived from equations 5.50 and 5.51. Each centroid image node has weights going to all the (linear) nodes in each vector field component layer. Each weight has the

[4] Not to be confused with their output values, which are also often denoted as x_i or y_i.

[5] The parameter α in these figures effectively determines the gain of the sigmoids, since it scales all weights. In this study it was set to 100.

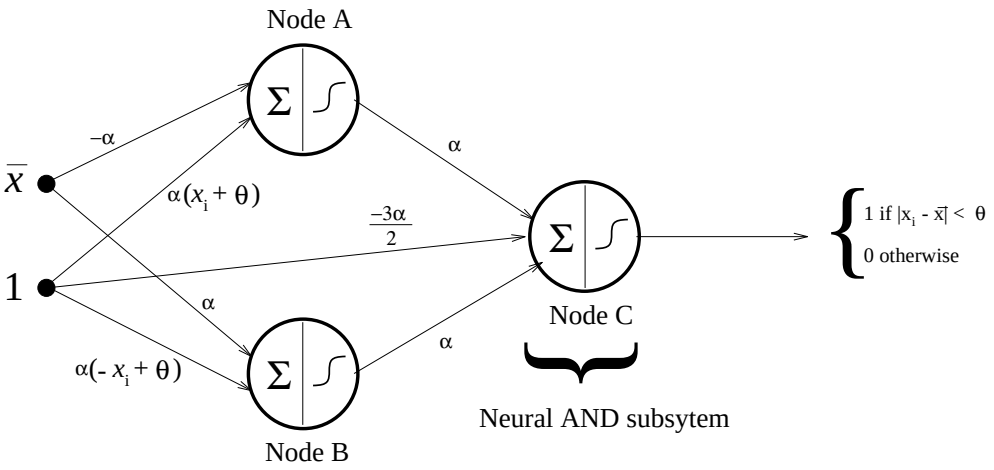


Figure 6.3: Coordinate matching module for node x_i. Output 1 when: $|x_i - \bar{x}| < \theta$.

value appropriate for the vector field at the destination node if the source node is the centroid. Since all nodes of the Centroid Image are zero except for the centroid node which is one, only the weights from the centroid node contribute to the vector field component images. Thus, weights corresponding to the vector fields for all possible centroid positions are present in the network, but only the appropriate ones contribute. Equation 6.5 shows the function computed by a node (k, l) in the rotation invariance vector field x-component image:

$$\begin{aligned} \mathrm{Rot}_{x_{(k,l)}} &= \sum_{i=0}^{x_{max}} \sum_{j=0}^{y_{max}} w_{(i,j)(k,l)} \mathcal{I}_{C_{(i,j)}} \\ w_{(i,j)(k,l)} &= \frac{-(j-l)}{\sqrt{(j-l)^2 + (i-k)^2}}. \end{aligned} \tag{6.5}$$

Note that it does not matter which node is chosen as the origin of coordinates in each layer when the weights are set, only that it is consistent between layers. Similar functions for setting weight values, derived from Equations 5.50 and 5.51, are used for the other three vector component images.

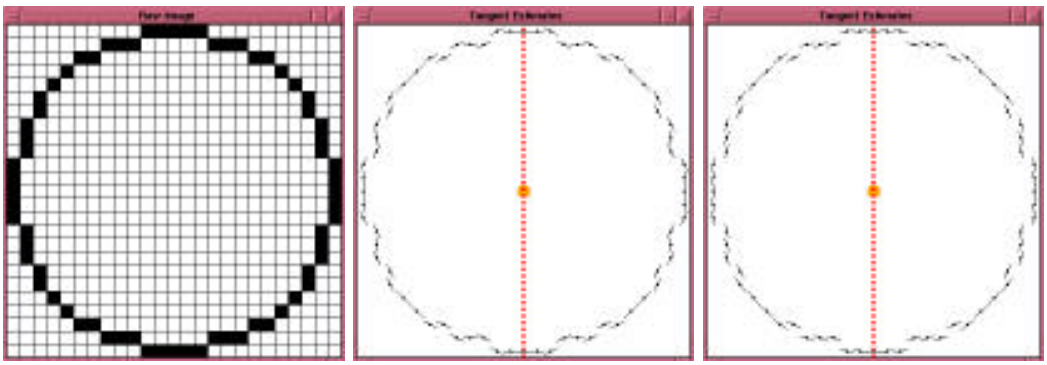
6.1.2 Local Orientation Extraction

A Tangent Estimate

In order to compute the Invariance Signature for an image, it is necessary to estimate the tangent vector at each point of the contour, since the mapping to invariance space requires the dot product of this with the three vector fields described in §5.2. A simple and robust estimate of the tangent vector at a point is the eigenvector corresponding to the largest eigenvalue of the covariance matrix of a square window centred on that

point.

The size of the window chosen defines the number of orientations possible. Figure 6.4 shows the orientation of the tangent estimates for an image of a circle for both 3×3 and 5×5 windows, calculated exactly. It is clear that the tangent estimates



(a) Original Image (b) 3×3 Window (c) 5×5 Window

Figure 6.4: Tangent estimation with varying window sizes, using the eigenvector corresponding to the largest eigenvalue of the covariance matrix.

calculated using the 5×5 window are better than those from the 3×3, but this comes at a computational cost, since the number of calculations is proportional to the square of the window size. Moreover, the "best" choice of window size depends upon the scale of the input contour. If the window size becomes large compared to the scale of the contour, the dominant eigenvector of the window's covariance matrix becomes a poor estimator of the tangent at a point, since the window is likely to include extraneous parts of the contour. This is clear from the tangent estimates shown in Figure 6.5, which was calculated with a window size of 35×35. Consequently, we choose to use a 3×3 window, both for its computational advantage and because it makes no assumptions about the scale of the input contour.

Discontinuities

Estimating this mapping presents some difficulties for a neural network, as it is not continuous. There are two discontinuities. The first arises when the dominant eigenvalue changes, and the orientation of the tangent estimate jumps by $\frac{\pi}{2}$ radians. The second is due to the flip in sign of the tangent vector when the orientation goes from π back to 0.

The first discontinuity can be avoided by using a weighted tangent estimate. Rather than being unity, we let the magnitude of the estimated tangent vector corresponds to strength of the orientation. Let the eigenvalues of the covariance matrix be λ_1 and

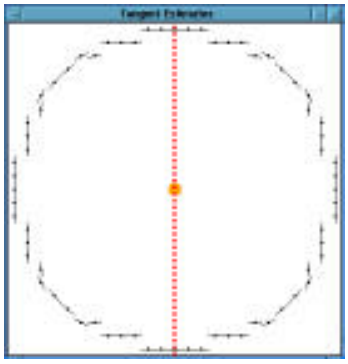


Figure 6.5: Tangents estimated with a window size of 35×35.

λ_2, where $\lambda_1 \geq \lambda_2$. The corresponding unit eigenvectors are $\hat{\mathbf{e}}_1$ and $\hat{\mathbf{e}}_2$. The weighted tangent vector estimate $\mathbf{s}$ is given by

$$\mathbf{s} = \begin{cases} \left(1 - \left|\frac{\lambda_2}{\lambda_1}\right|\right) \hat{\mathbf{e}}_1 & \text{if } \lambda_1 \neq 0, \\ 0 & \text{if } \lambda_1 = 0. \end{cases} \tag{6.6}$$

This weighting cause the magnitude of the estimated tangent vector to go to zero as $\left|\frac{\lambda_2}{\lambda_1}\right| \rightarrow 1$, and thus the $\frac{\pi}{2}$ jump is avoided. There is, however, still a discontinuity in the derivative.

Training a Neural Orientation Extraction Module (NOEM)

The aim, then, is to produce a neural network module to approximate this mapping. This module is to be replicated across the input layer so that the x and y components of the tangent vector are estimated everywhere. The result is two images of the same size as the input layer. The values are gated by the input image (used as dynamic weights), so that values are only transmitted for nodes corresponding to 1's in the input image. It was decided to produce this module by training a network on this task.

A training set was produced consisting of all 256 possible binary 3×3 input windows with a centre value of 1, and, for each, the two outputs given by Equations 6.6. A variety of networks was produced and trained on this task. The performance measure used, E, was the ratio of the sum squared error to a variance measure for the training set,

$$E = \frac{\sum_{c=1}^{N} \sum_{j=1}^{M} (t_{cj} - y_{cj})^2}{\sum_{c=1}^{N} \sum_{j=1}^{M} \left(\overline{t_{cj}} - t_{cj}\right)^2} \tag{6.7}$$

where N is the number of training exemplars, M the number of nodes in the output layer, t_{cj} the desired output value of the node and y_{cj} the actual output value.

A variety of standard, single hidden layer backpropagation networks was trained,

with different hidden layer sizes. It was found that the task could not be learnt without a significant number of hidden nodes, which is not unexpected for such a highly nonlinear function. The networks proved very unstable during training, so it was necessary to use an extremely small learning rate. As would be expected, the accuracy of the mapping improved with increasing hidden layer size, and the training time increased. The residual error did not stabilize, but continued to decrease steadily, albeit very slowly, as training was continued. The final accuracy achieved was thus ultimately determined by how long one was prepared to wait.

For the work reported in Squire and Caelli (1995), a three layer neural network with a 3×3 input layer, a 20 node hidden layer, and 2 output nodes was trained using backpropagation. After 6×10^6 iterations with a learning rate of 0.0005, a value of $E = 3.11\%$ was reached. This was considered sufficiently accurate for this module. Further investigations, as will be seen in §7.2, have indicated that even such seemingly small residual errors in the Local Orientation Extraction module can adversely affect final classification performance. Consequently, means of improving the accuracy of this module were sought.

Despite the universality of single hidden layer neural networks, several authors have indicated that networks with multiple hidden layers can be useful for extremely nonlinear problems [Sontag, 1992; Sarle, 1994]. Sarle (1994, p. 9) reports:

> Although a MLP with one hidden layer is a universal approximator, there exist various applications in which more than one hidden layer can be useful. Sometimes a highly nonlinear function can be approximated with fewer weights when multiple hidden layers are used than when only one hidden layer is used.

This suggested that a more compact solution might be obtained using a network with two hidden layers. Several standard four-layer networks were trained using backpropagation on this problem. It did indeed seem that they converged more rapidly, and to a smaller residual error. However, as with the single hidden layer networks, they were very sensitive to the learning rate, suggesting that perhaps a more sophisticated learning algorithm, such as QuickProp [Fahlman, 1988], would be more appropriate. After an initial large reduction, the error continued to decrease very slowly throughout training. The limitation on the final accuracy of the approximation appeared to be how long one was prepared to wait as much as the number of hidden nodes.

It is noted that this is a similar problem to edge extraction, the difference being that edge extraction is usually performed on images containing grey-scale gradients rather than on a binary, thin contour. Srinivasan, Bhatia and Ong (1994) have developed a neural network edge detector which produces a weighted vector output very much like the one described in Equation 6.6. They used an encoder stage which was trained competitively, followed by a backpropagation-trained stage. The encoder produced weight distributions closely resembling Gabor filters of various orientations and phases.

It is intended to try to produce a more compact and accurate tangent estimator using a MBNN incorporating a stage with Gabor weighting functions, as used in Caelli et al. (1993).

6.2 Calculation of the Local Measure of Consistency

The next stage of the network computes the Local Measure of Consistency of the tangent estimate at each point of the contour for each of the Lie vector fields. The output required is an *Invariance Image* for each Lie transformation. This is an image of the same size as the input image, where the value at each point is given by the absolute value of the dot produce of the estimated tangent vector to the contour and the Lie vector field at that point, as specified in Equation 5.34.

The neural implementation is simple. With reference to Figure 6.1, the x-component image from the Local Orientation Extractor provides dynamic weights to be combined with the x-component of each of the Lie vector fields. The same is done for the y-components. For each Lie transformation, there is thus layer of neurons which each have two inputs. The inputs come from the vector field images at the same coordinates, and are weighted by the Local Orientation images, again at the same coordinates. Note that points corresponding to zeros in the input image have tangent estimates of zero magnitude and thus make no contribution.

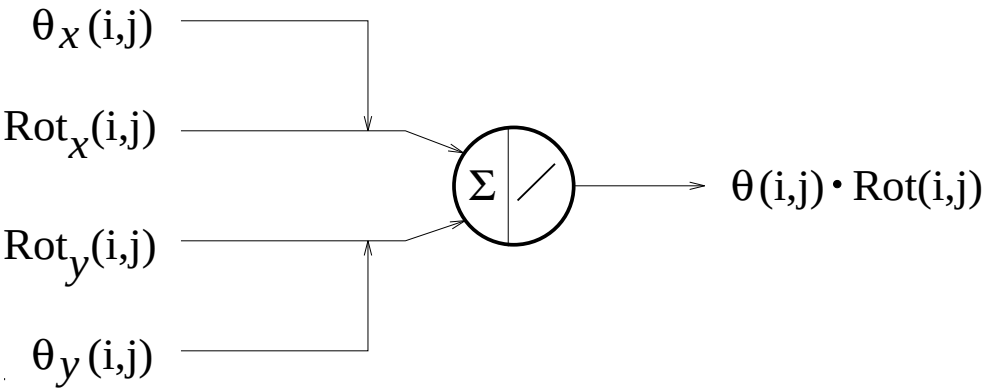


Figure 6.6: Calculation of the dot product of the tangent estimate θ and the vector field corresponding to rotation, for the image point at coordinates (i, j).

As an example, consider the subsystem corresponding to the point in the input image at coordinates (i, j). Figure 6.6 shows the neural implementation of the first stage of the calculation. Modules of this type are replicated for each point in the input image, and for each Lie vector field image. All that then remains is to pass the output of these modules through modules which calculate the absolute value of their input.

Figure 6.7 shows a neural module which calculates a good approximation of the

absolute value of its input. It is less accurate for inputs very close to zero, but in this application, where possible orientations are coarsely quantized, this does not cause any problems. This completes the neural calculation of the Local Measure of Consistency for each contour point with respect to each Lie vector field. All that remains is to combine these into an Invariance Signature.

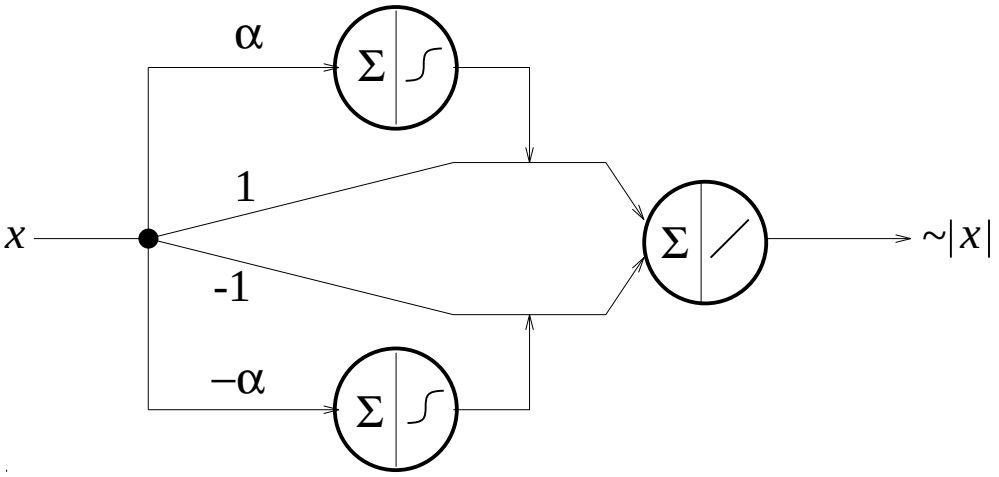


Figure 6.7: Neural module which calculates the absolute value of its input.

6.3 Calculation of the Invariance Signature

6.3.1 The Binning Unit

The ability of the MBNN approach to produce modules that perform tasks not usually associated with neural networks is illustrated by the binning unit shown in Figure 6.8. There is one of these modules for each of the n bins comprising the Invariance Signature histogram for each invariance class. Each binning module is connected to all the nodes in the Invariance Image, and inputs to it are gated by the binary input image, so that only the N nodes corresponding to ones in the input image contribute.

The n bins have width $\frac{1}{n-1}$, since the first bin is centred on 0, and the last on 1.[6] The system is designed so that Nodes A and B only have an output of 1 when the input x is within bin i. The condition for x to be in bin i is:

$$\frac{2i-1}{2(n-1)} < x < \frac{2i+1}{2(n-1)} \tag{6.8}$$

[6]This is because a bin ending at an extreme of the range would miss contributions from the extreme value since the edge of the neural bin is not vertical.

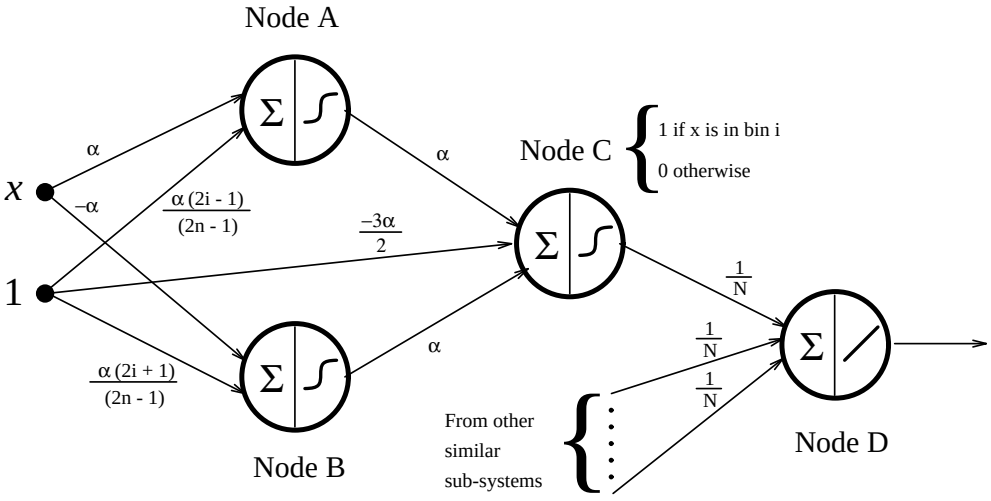


Figure 6.8: Neural binning module.

In order to detect this condition, the activations of Nodes A and B are set to:

$$net_A = \alpha x - \frac{\alpha(2i-1)}{2(n-1)} \tag{6.9}$$

$$net_B = -\alpha x + \frac{\alpha(2i+1)}{2(n-1)}. \tag{6.10}$$

The outputs of these nodes go to Node C, which computes a neural AND. Node D sums the contributions to bin i from all N nodes.

This concludes the calculation of the Invariance Signature. The end result is a signature which consists of $3n$ values, where n is the number of bins in the invariance signature histogram for each invariance class. This calculation has been achieved in the framework of a modular MBNN, where module functions are strictly defined, and are completely independent of any training data. The building blocks of the individual modules are restricted to simple summation artificial neurons, with either linear or sigmoidal transfer functions. The sole departure from standard neural network components is the introduction of dynamic weights, but, as already stated, these can be eliminated if product neurons are used, as in Higher Order Neural Networks (see §3.4.3). This shows the power of the modular approach to neural network design, and demonstrates that it is possible to design modules which perform tasks that are not often considered to be part of the neural networks domain, such as binning data.

A final module can be added to this network which is trained to classify patterns on the basis of the $3n$-dimensional Invariance Signature, rather than on the input patterns themselves. Classification performance will thus be guaranteed to be invariant under shift, rotation and scaling. The advantages of this approach are demonstrated in Chapter 7.

Chapter 7

Character Recognition with Invariance Signature Networks

7.1 Retention of Sufficient Information

In Chapter 5 a new class of invariant signature for two dimensional contours was derived. It was established that these Invariance Signatures are themselves invariant under rotations, dilations and translations of the contour. In Chapter 6 it was shown that a MBNN can be constructed which computes these signatures and classifies contours on that basis.

It remains now to demonstrate that these Invariance Signatures retain enough information content to be usable for pattern recognition, and that they are not unduly sensitive to noisy data such as that encountered in real applications. In order to do this, the system is applied to the classification of letters of the Roman alphabet, both for "perfect"[1] machine-generated training and test data, and for data gathered using physical sensors.

7.2 Perfect Data

7.2.1 Departures from Exact Invariance

Despite the fact that Invariance Signatures are provably invariant under rotations, dilations and shifts in the plane when calculated for a continuous contour, departures from invariance occur in real applications in several ways. Scanned data contains random noise from the sensor, though the quality of scanners now available renders this negligible for this application. More important sources of error are discussed below.

[1] Throughout this chapter, the term "perfect" will be used to describe data which is both noise-free and unambiguous.

Quantization Noise Noise is introduced into the tangent estimation procedure by the sampling of the contour. Since the estimated tangent orientation is quantized,[2] the value of the Local Measure of Consistency ι_G can be changed when a contour is quantized at a new orientation. It is possible to compensate partially for this effect by using sufficiently wide bins when calculating the Invariance Signature $I(\iota_G)$, but there will still be errors when the change in estimated orientation moves the resultant ι_G across bin boundaries.

Ambiguous Characters In many fonts some letters are rotated or reflected versions of other letters, such as {b, d, p, q} and {n, u} in many sans serif fonts. In fonts with serifs, the serifs are often so small that they make a negligible contribution to the Invariance Signature. Consequently, it is impossible to classify isolated characters into 26 classes if shift, rotation, scale and reflection invariance is desired. Admittedly, reflection invariance is not usually desired in a character recognition system, but it is an unavoidable characteristic of the ISNNC. In commercial optical character recognition systems, context information is used to resolve ambiguous letters,[3] which occur even in systems without inherent invariances. Such an approach would be equally applicable as a post-processing stage for the ISNNC system.

7.2.2 The Data Set

These effects can be avoided by using a computer to produce a "perfect" data set, which is free from quantization noise and contains no letters which are transformed version of others in the data set. Such a data set can be used to demonstrate that Invariance Signatures retain sufficient information for classification in the absence of noise, and these results can be used as a basis for assessing the performance of the system on real data.

A training data set was created using a screen version of the Helvetica font. Only the letters {a, b, c, e, f, g, h, i, j, k, l, m, n, o, r, s, t, v, w, x, y, z} were used, so that ambiguity was avoided. An 18×18 binary image of each letter was produced. This training data set is shown in Figure 7.1.

A "perfect" test data set was created by computing reflected and rotated versions of the training data, where all rotations were by multiples of $\frac{\pi}{2}$ radians, so that there was no quantization error. This test data set is shown in Figure 7.2.

[2] See §6.1.2.

[3] *i.e.* surrounding letter classifications and a dictionary of acceptable words.

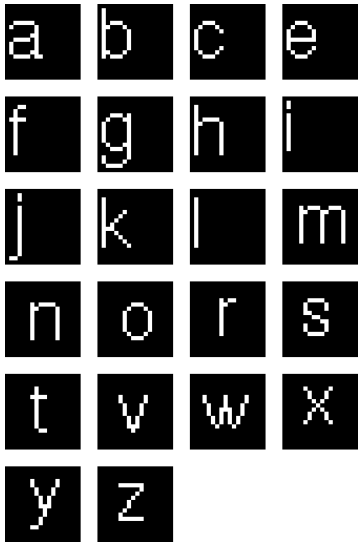


Figure 7.1: Training set of canonical examples of unambiguous characters.

7.2.3 Selected Networks Applied to this Problem

Simulations indicated that the training set shown in Figure 7.1 could be learnt by a neural network with no hidden layer: it is a linearly separable problem [Minsky and Papert, 1969]. It was assumed that this would also be true of the Invariance Signatures calculated from these data.[4] Consequently, two different network architectures were constructed for comparison.[5] The first was a TNN with a 18×18 input layer, no hidden layers, and a 1×22 output layer. The other was an ISNNC, with an 18×18 input layer for the Invariance Signature calculation stage. The Invariance Signature was calculated using 5 bins for each Lie transformation, meaning that the Invariance Signature layer had 3×5 nodes. This was connected directly to a 1×22 output layer, with no intervening hidden layers, forming an linear classifier sub-network.

7.2.4 Reduction of Data Dimensionality

It should be noted that although the Invariance Signature Calculation stage of the ISNNC has to be run every time a new pattern is to be classified at run time, it is only necessary to calculate the Invariance Signatures of the training and test data once. The final classification stage of the ISNNC can then be trained as a separate module. This can lead to an enormous reduction in the time taken to train a network, since the number of training passes required to learn a large and complex data set is typically

[4] This was confirmed by subsequent experiments.

[5] All networks described were constructed, trained and evaluated using the **Xnet** simulator described in Appendix B.

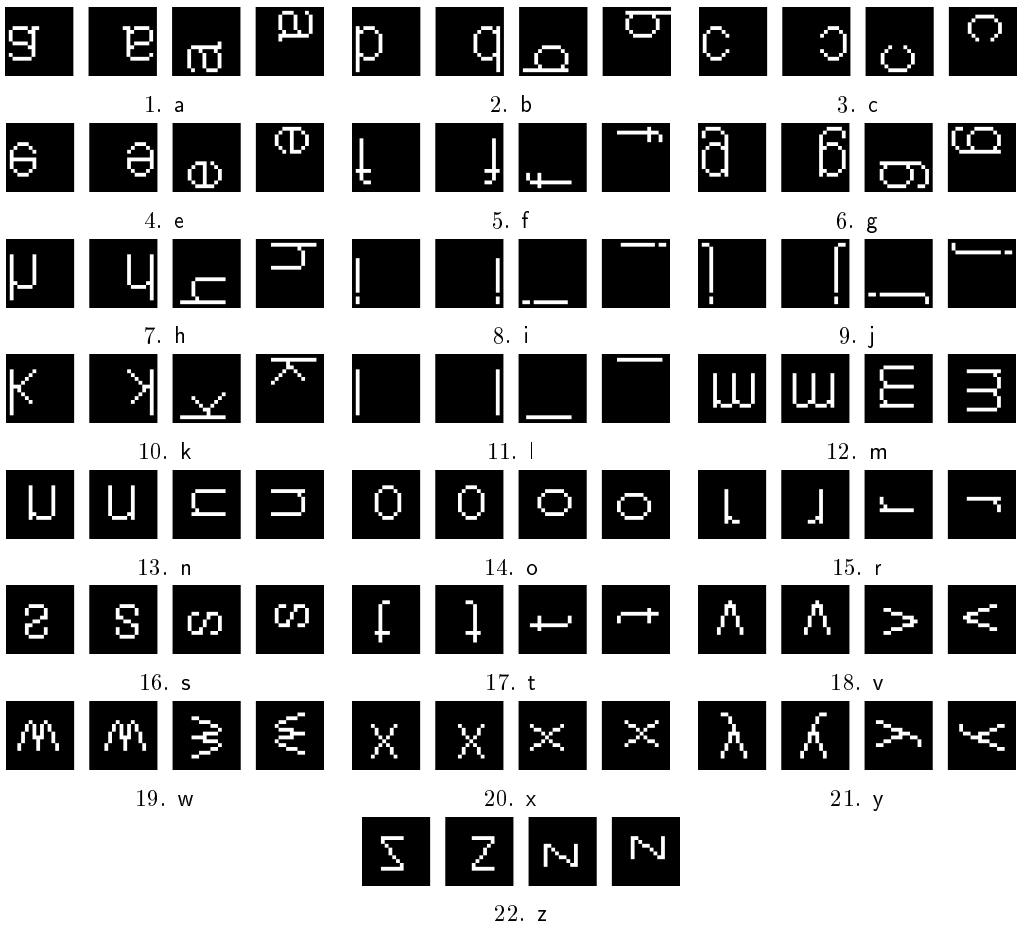


Figure 7.2: Test set of ideally shifted, rotated and reflected letters.

order 10^2 to 10^3, and the total number of parameters n_p in a TNN is:

$$n_p = \sum_{i=1}^{N-1} (\text{nodes in layer})_{i-1} \times (\text{nodes in layer})_i \tag{7.1}$$

where N is the total number of layers, and i is the layer number, where the input layer is labeled 0. The iteration time during training is proportional to n_p, so, for this example, each training iteration for the Invariance Signature classification stage will be $\frac{18\times 18}{3\times 5} = 21.6$ times faster than that for the TNN.

The calculation of the Invariance Signatures is admittedly time-consuming, but the time is made up many times over during training when the input image size is large. In real applications, the image size is typically larger than the 18×18 image used in this demonstration. ISNNC systems can therefore provide an extremely significant reduction in the dimensionality of the training data, and a corresponding reduction in the time taken to train the classification network. Moreover, the simulation of the ISNNC on a sequential computer is unable to take advantage of the inherently

parallel, local computations that characterize many of the ISNNC modules. A parallel implementation would largely alleviate the time-consuming nature of the Invariance Signature calculation.

7.2.5 Perfect and Network-Estimated Local Orientation

Since the neural network implementation of the Local Orientation Unit described in §6.1.2 still had some residual error, two versions of the Invariance Signature training and test data were created, one with the local orientation at each point calculated directly from the covariance matrix eigenvectors, and the other using the neural network module. Results from these data sets will be compared to evaluate the importance of accurate orientation extraction in the neural module.

Ten examples of each network were made, each with a different initialization of parameters, so that it could be ensured that the results were reproducible. The TNNs and the ISNNC linear classifier modules were all trained using the backpropagation algorithm, with the weight update equation shown in Equation 4.14. The parameters were $\epsilon = 0.5$ and $\alpha = 0.5$ for the Invariance Signature classification modules, and $\epsilon = 0.025$ and $\alpha = 0.5$ for the TNNs. The difference in ϵ is to compensate for the different fan-in to the output nodes in the two networks. Training was continued for 1000 iterations. A network was deemed to have classified a pattern as belonging to the class corresponding to the output node which had the highest output value, so no patterns were rejected.

Results for Traditional Neural Networks

The results obtained with TNNs are summarized in Table 7.1. It is clear that the TNNs do not exhibit transformation invariance. They could not be expected to do so, since no transformed versions of the patterns were included in the training data.

It might be expected that the fact that the final performance of the TNNs is better than chance ($\frac{1}{22} = 4.5454\%$) is because some of the transformations of the training data resulted in patterns almost identical to the untransformed training pattern for some of the highly symmetrical letters in the training set (*e.g.* o, s, x and z). Analysis of which test patterns were classified correctly, however, shows that this is not the case. The 12 correctly classified test patterns are shown in Figure 7.3. No reason for these particular patterns being classified correctly is obvious. It is well-known that the "rules" generated by TNNs are hard to extract, although progress has been made in that area [*e.g.* Wiles and Ollila, 1993; Wiles and Elman, 1995]. What is clear, however, is that chance must play a part in some of these classifications, for instance i4, which shares no "on" pixels with the training example of i (see Figures 7.1 and 7.3).

Table 7.2 shows that marginally better performance on the test data could have

	Best Performance		Final Performance	
	Training Data	Test Data	Training Data	Test Data
Network 1	100.00	14.773	100.00	13.636
Network 2	100.00	14.773	100.00	13.636
Network 3	100.00	15.909	100.00	13.636
Network 4	100.00	14.773	100.00	13.636
Network 5	100.00	14.773	100.00	13.636
Network 6	100.00	15.909	100.00	13.636
Network 7	100.00	14.773	100.00	13.636
Network 8	100.00	14.773	100.00	13.636
Network 9	100.00	14.773	100.00	13.636
Network 10	100.00	14.773	100.00	13.636
$\mu \pm \sigma$	100.00 ± 0.000	15.000 ± 0.454	100.00 ± 0.000	13.636 ± 0.000

Table 7.1: Classification performance (% correct) of traditional neural network classifiers trained for 1000 iterations with the data shown in Figure 7.1 and tested with the "perfect" data set in Figure 7.2.

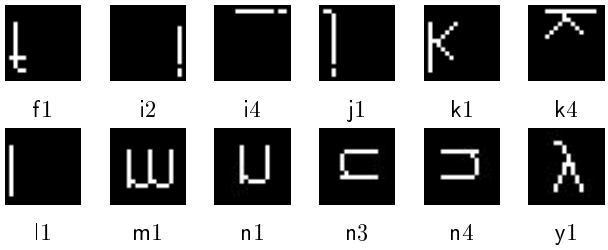


Figure 7.3: Test patterns classified correctly by the TNNs.

been achieved by employing an early-stopping scheme in which training was stopped when classification performance on the test set started to deteriorate. This would, however, have been at the cost of less than 100% correct performance on the training data. It should be noted that the error on the test set, as defined in Equation 1.3, decreased throughout the entire 1000 iterations. This indicates that an early-stopping scheme based on the test set error would fail to improve classification performance in this case. Figure 7.4 shows how the classification performances and errors typically varied during training.

It should be acknowledged that the TNNs could not really be expected to perform better than chance on this test set given the training set used. Their architecture provides no invariances, and generalization cannot be expected unless multiple transformed versions of the patterns are included in the training set. This argument can be used against all the comparisons between TNNs and MBNNs in this thesis: they are not really fair. Nevertheless, these comparisons between naive applications of TNNs and specifically-designed MBNNs demonstrate that MBNNs can perform successfully using training sets completely inadequate for TNNs. Moreover, these MBNNs are of

		% Correctly Classified	
	Iteration	Training Data	Test Data
Network 1	86	100.00	14.773
Network 2	33	95.455	14.773
Network 3	1	90.909	15.909
Network 4	85	100.00	14.773
Network 5	94	100.00	14.773
Network 6	1	90.909	15.909
Network 7	95	100.00	14.773
Network 8	1	95.455	14.773
Network 9	86	100.00	14.773
Network 10	19	95.455	14.773
$\mu \pm \sigma$	50 ± 40	96.818 ± 3.550	15.000 ± 0.454

Table 7.2: Performance at iteration at which best performance on test data occurred for traditional neural network classifiers trained with the data shown in Figure 7.1 and tested with the "perfect" data set in Figure 7.2.

lower dimensionality than the TNNs. Providing with the TNNs with sufficiently large training sets would only make their training still more computationally-expensive, with no *guarantee* of invariant performance.

Results with Perfect Local Orientation

The results for the ten ISNNCs which used perfect Local Orientation Extraction are shown in Table 7.3. The average number of iterations for 100% correct classification to be achieved was 220. Since the problem is linearly-separable, it could in fact be solved directly, using a linear equations technique such as singular-valued decomposition [Press et al., 1992]. Since weights may set by any method at all in the MBNN paradigm, this makes the comparison of convergence times somewhat irrelevant. Nevertheless, the MBNN modules, although taking, on average, 4.4 times as many iterations to converge as the TNNs, were still 4.9 times faster to train, due to their lower dimensionality, as discussed in §7.2.4.

As expected, the ISNNCs generalize perfectly to the transformed images in the test set. The network architecture constrains the system to be shift- rotation-, scale- and reflection-invariant in the absence of quantization noise, so this is no surprise. Importantly, the result indicates that sufficient information is retained in the 5 bin Invariance Signatures for all 22 unambiguous letters of the alphabet to be distinguished. Inspection of the percentage sum squared error values after each iteration indicated that the error on the test set was indeed identical to that on the training set: for perfect data, the ISNNC produces perfect results.

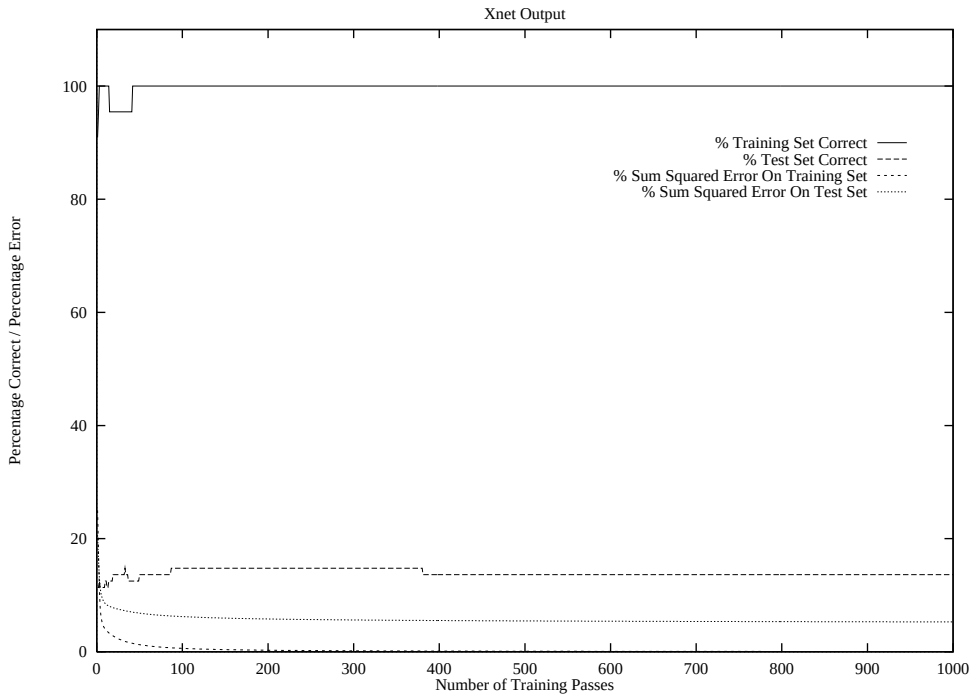


Figure 7.4: Classification performance and errors of TNN 2 during training.

Results with Network-Estimated Local Orientation

The perfect results above were obtained using a hybrid system, which used a module which was not a neural network to calculated the local orientation at each point. The neural module for local orientation extraction used here was trained using backpropagation until 98.93% of the variance in the training data was explained (see Equation 6.7). The Invariance Signatures for the test and training data were calculated using the system incorporating this module, and the classification module was then trained separately using these Invariance Signatures. Systems were produced with both 5 and 10 bin Invariance Signatures. The results obtained are shown in Tables 7.4 and 7.5.

The misclassified patterns for the 5 bin ISNNCs were those shown in Figure 7.5. `t1` and `t2` were classified as `f`, which is understandable, since there is very little difference indeed between the patterns. `t4` was misclassified as `a`. All ten networks had these same misclassifications.

These data show that small residual error in the neural Local Orientation Extraction module does cause a degradation in the transformation invariant classification performance of the ISNNCs. These results are, however, still far superior to those for the TNNs. Moreover, there is no reason that the Local Orientation Extraction module could not be trained further. It is to be expected that the invariant classification performance would continue to approach 100% as the accuracy of the module was improved.

	Best Performance		Final Performance	
	Training Data	Test Data	Training Data	Test Data
Network 1	100.00	100.00	100.00	100.00
Network 2	100.00	100.00	100.00	100.00
Network 3	100.00	100.00	100.00	100.00
Network 4	100.00	100.00	100.00	100.00
Network 5	100.00	100.00	100.00	100.00
Network 6	100.00	100.00	100.00	100.00
Network 7	100.00	100.00	100.00	100.00
Network 8	100.00	100.00	100.00	100.00
Network 9	100.00	100.00	100.00	100.00
Network 10	100.00	100.00	100.00	100.00
$\mu \pm \sigma$	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00

Table 7.3: Classification performance (% correct) of Invariance Signature Neural Network Classifiers (with perfect Local Orientation Extraction) trained for 1000 iterations with the data shown in Figure 7.1 and tested with the "perfect" data set in Figure 7.2.

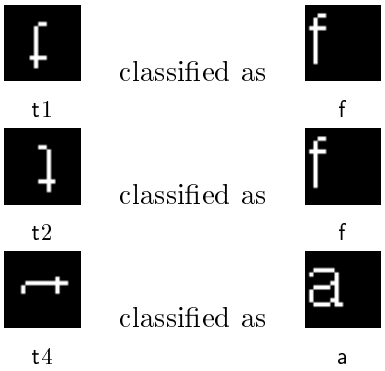


Figure 7.5: Test Patterns Misclassified by the 5 Bin Invariance Signature Neural Network Classifiers, and the training examples as which they were incorrectly classified.

The results for the 10 bin system in Table 7.5 show that the effects of inaccuracies in the Local Orientation Extraction module are greater when the number of bins is increased. This is due to the fact that the errors can cause the consistency measure at a point to change bins more easily this way, thus altering the histogram Invariance Signature.

7.3 Optical Character Recognition

Having demonstrated that Invariance Signatures retain sufficient information for the classification of "perfect" character data, it is now necessary to show that the system can be used to achieve transformation invariant recognition of data in the presence of sensor and quantization noise. To this end, it was decided to apply ISNNCs to the classification of scanned images of shifted and scaled printed alphabetic characters.

	Best Performance		Final Performance	
	Training Data	Test Data	Training Data	Test Data
Network 1	100.00	96.591	100.00	96.591
Network 2	100.00	96.591	100.00	96.591
Network 3	100.00	96.591	100.00	96.591
Network 4	100.00	96.591	100.00	96.591
Network 5	100.00	96.591	100.00	96.591
Network 6	100.00	96.591	100.00	96.591
Network 7	100.00	96.591	100.00	96.591
Network 8	100.00	96.591	100.00	96.591
Network 9	100.00	96.591	100.00	96.591
Network 10	100.00	96.591	100.00	96.591
$\mu \pm \sigma$	100.00 ± 0.00	96.591 ± 0.00	100.00 ± 0.00	96.591 ± 0.00

Table 7.4: Classification performance (% correct) of 5 Bin Invariance Signature Neural Network Classifiers (with neural Local Orientation Extraction) trained for 1000 iterations with the data shown in Figure 7.1 and tested with the "perfect" data set in Figure 7.2.

7.3.1 The Data Set

The data set used was derived from the set of shifted and rotated versions of the lowercase Roman alphabet shown in Figure 7.6. Each character appears in 18 different orientations, at rotation increments of 20 degrees. The shifts arise due to the fact that characters are extracted using a bounding-box technique which takes no account of the centroid position.

An A4 page with these characters printed onto to it from a laser printer at 300 dots per inch was scanned at 75 dots per inch using a UMAX Vista-S6 Scanner. The connected regions in this image were then detected, and the minimum bounding-box which could contain the largest of the characters was calculated (56 × 57 pixels). The image was then segmented into separate characters, and each character was thinned using an algorithm due to Chen and Hsu (1988). The thinning was necessary since the Invariance Signature method is only applicable to thin contours. This set of 468 characters was partitioned into a training set and a test set. The training set consisted of the characters rotated by angles in the range $[0°, 160°]$ relative to the upright characters, and the test set of those in the range $[180°, 340°]$. In Figure 7.6, those characters to the left of the dashed line were used for training, and those to the right for testing. The resultant scanned, extracted and thinned training set is shown in Figures 7.7 and 7.8, and the test set in Figures 7.9 and 7.10, respectively.

It is interesting to compare the subjective visual similarity between the Invariance Signatures both within and between classes for these extracted data. Figure 7.11 shows the first four training examples of the letter "a", accompanied by images showing the tangent estimates at each point of the contours. The tangent estimate is represented

	Best Performance		Final Performance	
	Training Data	Test Data	Training Data	Test Data
Network 1	100.00	95.455	100.00	93.182
Network 2	100.00	95.455	100.00	93.182
Network 3	100.00	95.455	100.00	93.182
Network 4	100.00	95.455	100.00	93.182
Network 5	100.00	95.455	100.00	93.182
Network 6	100.00	95.455	100.00	93.182
Network 7	100.00	95.455	100.00	93.182
Network 8	100.00	95.455	100.00	93.182
Network 9	100.00	95.455	100.00	93.182
Network 10	100.00	95.455	100.00	93.182
$\mu \pm \sigma$	100.00 ± 0.00	96.591 ± 0.00	100.00 ± 0.00	96.591 ± 0.00

Table 7.5: Classification performance (% correct) of 10 Bin Invariance Signature Neural Network Classifiers (with neural Local Orientation Extraction) trained for 1000 iterations with the data shown in Figure 7.1 and tested with the "perfect" data set in Figure 7.2.

by a line segment with the orientation of the estimated tangent. These images do not indicate the weight assigned to each estimated tangent.

It is not easy to interpret the similarity between these tangent representations of the contours. For this, it is necessary to see the Invariance Signatures. Figure 7.12 shows the Invariance Signatures for the patterns in Figure 7.11 with respect to rotation, dilation and translation. The Invariance Signatures are represented by the Invariance Measure Density histograms. It is immediately apparent that there is a great deal of (subjective) similarity between these representations of the shifted and rotated patterns. They are not identical, as a result of the various sources of noise discussed in §7.2. For classification purposes, however, all that is required is that these signatures be more similar to each other within a letter class than between letter classes. This must be determined by experiment.

Figures 7.13 and 7.14 show the equivalent data for the first four training patterns for the letter "x". These again show the marked within-class similarity, and are also distinctly different to the signatures for the letter "a": these letters were deliberately chosen since "a" is "quite rotationally-invariant", whereas "x" is "quite dilationally-invariant".

7.3.2 Selected Networks Employed for this Problem

The methodology employed for this experiment was identical to that used for the synthetic data, described in §7.2.3. The TNN used in this case had a 56×57 node input layer, and a 1×26 output layer, giving a massive 83018 independent parameters to be estimated. With only 234 training patterns and 83018 parameters, this problem

is almost certain to be linearly separable. Simulation verified that this problem was indeed learnable by a network with no hidden layer.

A variety of different classification modules was tried, for ISNNCs with both 5 and 10 bin Invariance Signatures. These included the simple linear classifier, and a variety of MLP classifiers with differing numbers of nodes in their hidden layers. These experiments indicated that 5 bin Invariance Signatures were insufficient for this problem, and that it was not linearly separable. It also became clear that the errors introduced by the slightly inaccurate network-estimated Local Orientation Extraction caused a noticeable departure from invariance (see §7.2.5). For these reasons, the results presented are for 10 bin ISNNCs with directly-calculated ("perfect") Local Orientation and a MLP classification module. The MLP classifier had a 3×10 node input layer, a 15 node hidden layer, and a 26 node output layer. This classifier has only 881 parameters to be estimated, a reduction of 99% compared to the TNN linear classifier. This translates to a dramatic reduction in both the storage space and the training time required. Only five TNNs were trained, partly because of the training time needed, and partly because the results were so consistent.

7.3.3 Results for Traditional Neural Networks

It might have been expected that the TNNs would perform better on this task than on the task with the synthetic data described in §7.2.3, since this training set does indeed contain differently transformed versions of the canonical untransformed characters. As can be seen from Table 7.6, the results are in fact similar, and slightly worse in this case (average best percent correct of 13.932 ± 0.300 compared with 15.000 ± 0.454). Although better than chance (3.85% correct), these results are no where near usable for an optical character recognition system.

	Best Performance		Final Performance	
	Training Data	Test Data	Training Data	Test Data
Network 1	100.00	13.675	100.00	11.111
Network 2	100.00	14.103	100.00	11.111
Network 3	100.00	14.103	100.00	10.684
Network 4	100.00	14.103	100.00	10.684
Network 5	100.00	13.675	100.00	11.111
$\mu \pm \sigma$	100.00 ± 0.00	13.932 ± 0.300	100.00 ± 0.00	10.940 ± 0.209

Table 7.6: Classification performance (% correct) of Traditional Neural Network Classifiers trained for 200 iterations with the data shown in Figures 7.7 and 7.8 and tested with data set in Figures 7.9 and 7.10.

7.3.4 Results for Invariance Signature Neural Network Classifiers

The results obtained with ISNNCs attempting to classify the characters into 26 classes appear in Table 7.7. These results show that the ISNNCs achieve a much higher correct classification rate on the test set than the TNNs. The failure of the ISNNCs to achieve 100% correct classification of the training set is in fact not surprising. The test and training sets used for this problem have each individual character of the alphabet mapped to a separate class. Yet, as was discussed in §7.2, the sets of characters {b, d, p, q} and {n, u} are identical under rotations and reflections: transformations under which the ISNNC output is invariant. The expected training set performance for noise-free data is thus $100 \times (20 + 0.25 \times 4 + 0.5 \times 2)/26 = 85\%$ correct. Any performance on the training data better than this must be the result of the fitting of noise in the training data.

	Best Performance		Final Performance	
	Training Data	Test Data	Training Data	Test Data
Network 1	95.726	72.650	95.726	71.795
Network 2	96.154	72.222	96.154	70.940
Network 3	96.154	73.077	96.154	69.658
Network 4	95.299	73.932	95.299	70.513
Network 5	94.872	71.795	94.872	71.368
Network 6	95.726	72.650	95.726	68.803
Network 7	96.154	73.504	96.154	69.658
Network 8	97.436	72.222	97.436	70.513
Network 9	94.444	74.359	94.444	69.658
Network 10	96.154	70.940	96.154	70.940
$\mu \pm \sigma$	95.812 ± 0.783	72.735 ± 0.971	95.812 ± 0.783	70.385 ± 0.877

Table 7.7: Classification performance (% correct) of 10 bin Invariance Signature Neural Network Classifiers (with perfect Local Orientation Extraction) trained for 40000 iterations with the data shown in Figures 7.7 and 7.8 and tested with data set in Figures 7.9 and 7.10.

In order to assess the magnitude of this effect, training and test sets were created in which {b, d, p, q} were assigned the same label, as were {n, u}. The results are shown in Table 7.8. These results show that the average final test set performance was improved by 14.8% by this re-labeling, which is very close to the maximum possible 15.4% achievable if this were the only source of error.

The residual difference between training and test set error is generalization error, rather than invariance error. These networks were trained with only 9 examples of each character, and these examples are quite noisy. There is the unavoidable quantization noise, but there are also some quite marked artifacts, such as the loops introduced by the thinning algorithm, one of which can be seen in pattern a03 in Figure 7.12, and in several patterns in both the training and test sets in Figures 7.7 though 7.10. There

	Best Performance		Final Performance	
	Training Data	Test Data	Training Data	Test Data
Network 1	98.718	87.179	98.718	87.179
Network 2	99.145	83.761	99.145	82.906
Network 3	99.573	86.752	99.145	85.897
Network 4	99.573	86.752	99.145	85.043
Network 5	98.718	88.034	98.718	88.034
Network 6	99.573	85.897	99.145	85.470
Network 7	100.00	85.897	100.00	83.761
Network 8	98.291	86.325	98.291	85.043
Network 9	97.863	86.752	97.863	85.043
Network 10	99.145	84.615	99.145	83.761
$\mu \pm \sigma$	99.056 ± 0.628	86.196 ± 1.179	99.056 ± 0.628	85.214 ± 1.483

Table 7.8: Classification performance (% correct) of 10 bin Invariance Signature Neural Network Classifiers (with perfect Local Orientation Extraction) trained for 10000 iterations with the data shown in Figures 7.7 and 7.8 and tested with data set in Figures 7.9 and 7.10, modified to label characters which can be transformed into each other as the same character.

are also two erroneous test patterns: the result of clipping in the segmentation process. They are the final m and the final w. These were left in, as such errors can and do occur in practical applications.

Misclassifications						
d → n	f → t	f → j	i → l	i → l	i → l	i → l
i → l	i → l	j → r	k → f	k → r	k → f	l → i
m → h	n → b	n → b	q → n	r → f	r → y	r → k
r → i	t → f	t → f	t → f	t → f	u → h	w → m

Table 7.9: Failure analysis for Network 5 from Table 7.8.

The errors made by the ISNNCs are not random. To illustrate this, a failure analysis is presented for Network 5 from Table 7.8, showing how the test patterns were misclassified. Patterns which are perceptually similar are responsible for many of the misclassifications. This means that prior information about likely errors could be used in conjunction with these classifications to aid error correction. This failure analysis indicates that the ISNNCs often appear to make errors that are very "human", which is a promising indication that the Invariance Signature measures contour similarity in a way similar to the (unknown) measure used by humans. Inspection of the actual thinned patterns used for training and testing the networks indicates that the patterns for the letters {i, j, l} are little more than straight lines. If these were to be re-labeled as the same character, final test set performance for Network 5 would improve to 91.026% correct. If the same were done for {f, t}, which are also extremely similar, test set performance would be 93.162% correct.

The misclassifications of patterns to the classes b and n may be due to the fact that the re-labeled dataset implies a non-uniform prior probability distribution for these classes: b was used as the target class for input patterns corresponding to {b, d, p, q}, and thus occurs four times more frequently in the training data than any other class. Similarly, n was used as the target for inputs {n, u}. As discussed in §3.2.6, networks will tend to "guess" these classes more frequently than the others.

In the context of the extremely small training set, this is a remarkable result, comparable with character recognition results achieved by others with thousands, rather than tens, of training patterns. Such re-relabeling is in any case an essential feature of a *truly* invariant optical character recognition system, since some characters are inherently ambiguous under rotation and reflection. Others are extremely similar, and noise can render them indistinguishable. In any real optical character recognition system, a dictionary is used to verify recognized words, and context is used to correct incorrectly labeled patterns. Another possibility is that the orientation of correctly-classified unambiguous characters could be used to infer the correct labeling of ambiguous characters.

Perhaps most importantly, these results show that ISNNCs can classify correctly patterns which have been transformed by arbitrarily large amounts. The shifts and rotations of the input images are not restricted to small perturbations of the example patterns. This indicates that the ISNNCs are performing truly invariant pattern recognition, rather than interpolation-based generalization.

This study should be considered to be a "proof of concept", both for the Invariance Signature as a contour descriptor, and for MBNNs which classify on that basis. It is not intended to be a large-scale experiment which corresponds to a real application. Such a study would require much greater quantities of data than are used here. We believe, however, that the experiments presented here are sufficient to demonstrate that the theoretical work of Chapters 5 and 6 has resulted in a system that is genuinely useful for robust invariant pattern recognition. The quality of the results obtained is in fact remarkable when the size of the training sets is compared to those used in other neural network approaches to the optical character recognition problem, which frequently consist of many thousands of input-output examples.

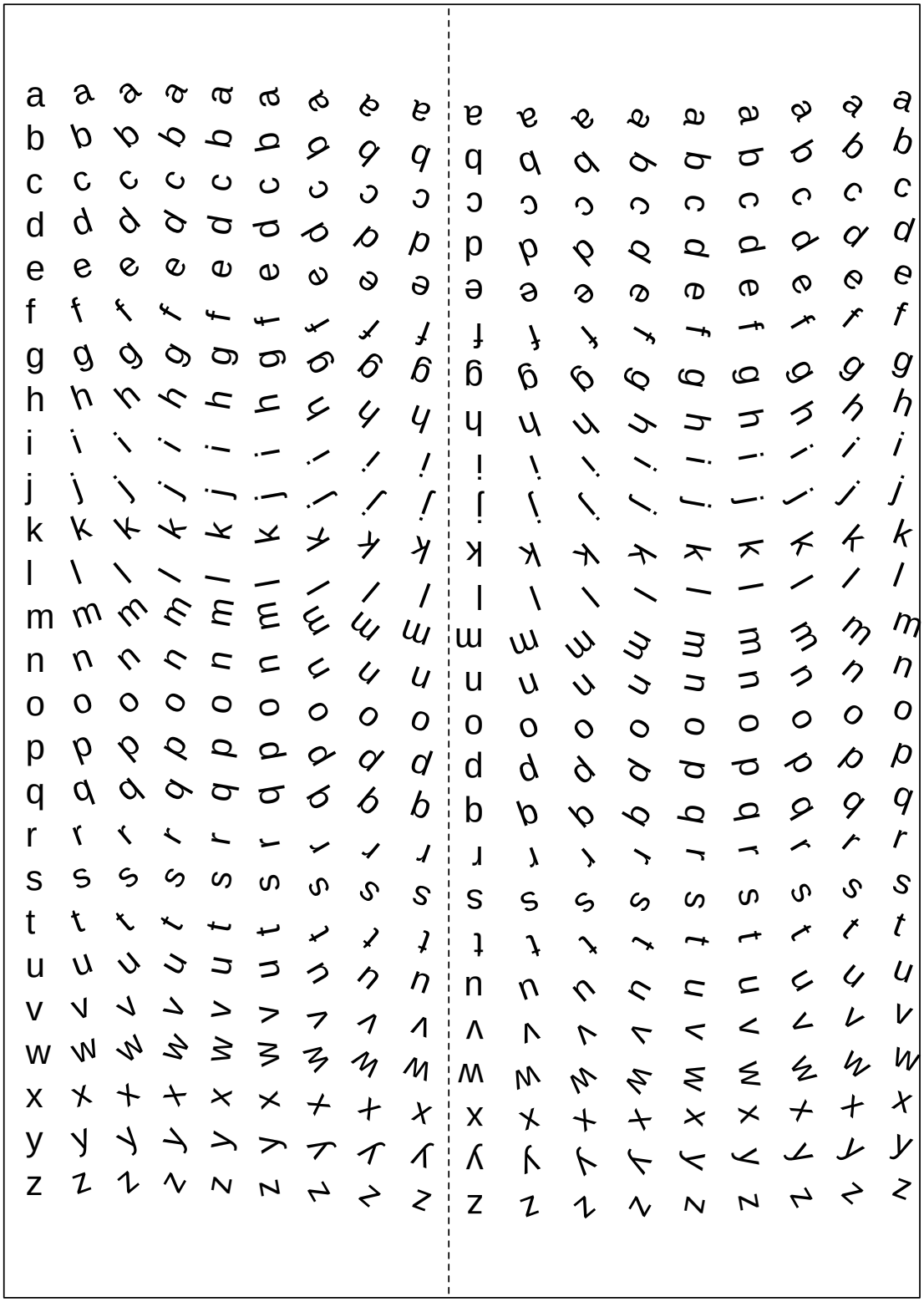


Figure 7.6: The characters scanned to generate the real data set. The box shows the border of an A4 page (210mm × 297mm) so that the size of the original characters may be judged. The dashed line shown the partition into training and test data.

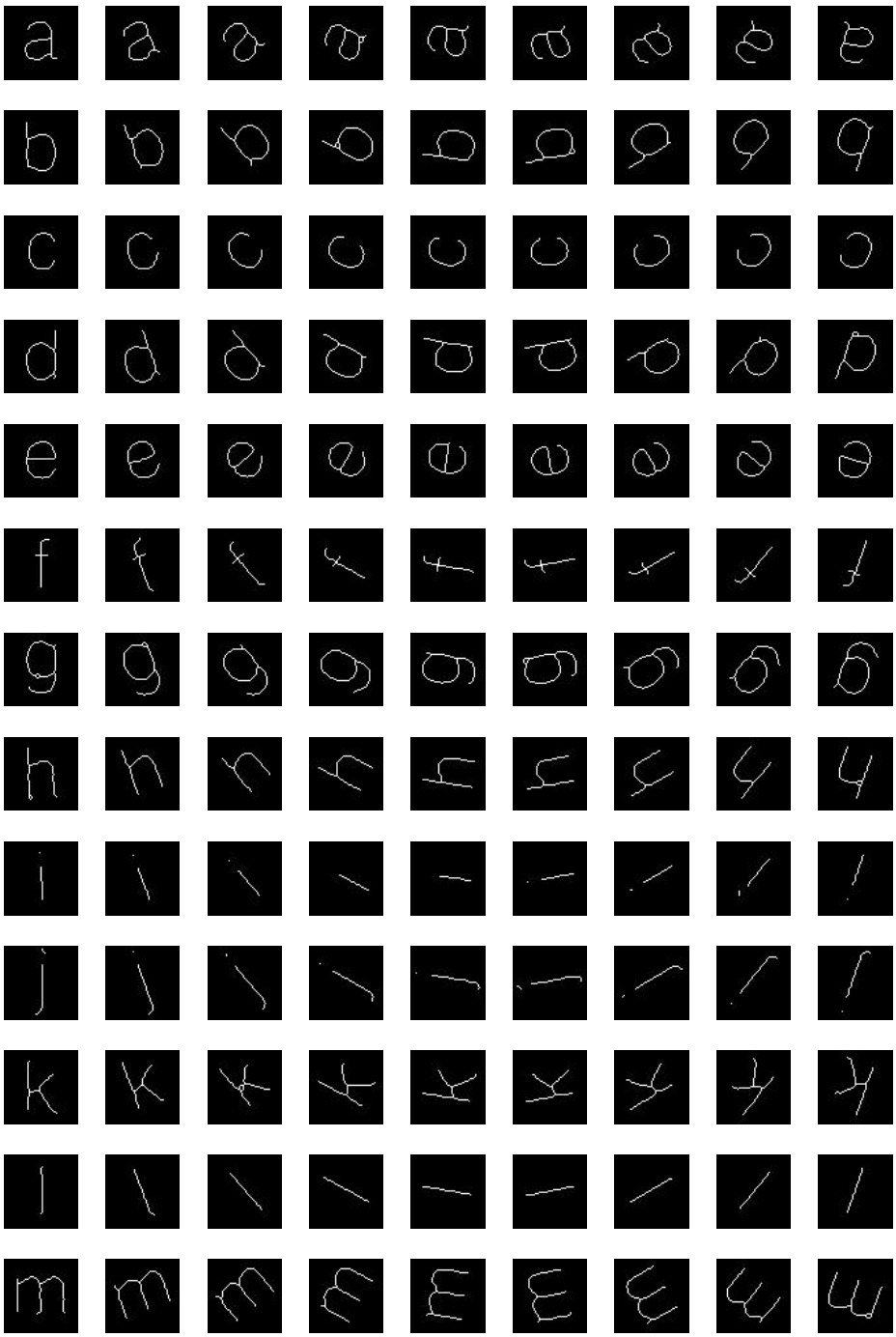


Figure 7.7: Training set of thinned, scanned characters (Part 1).

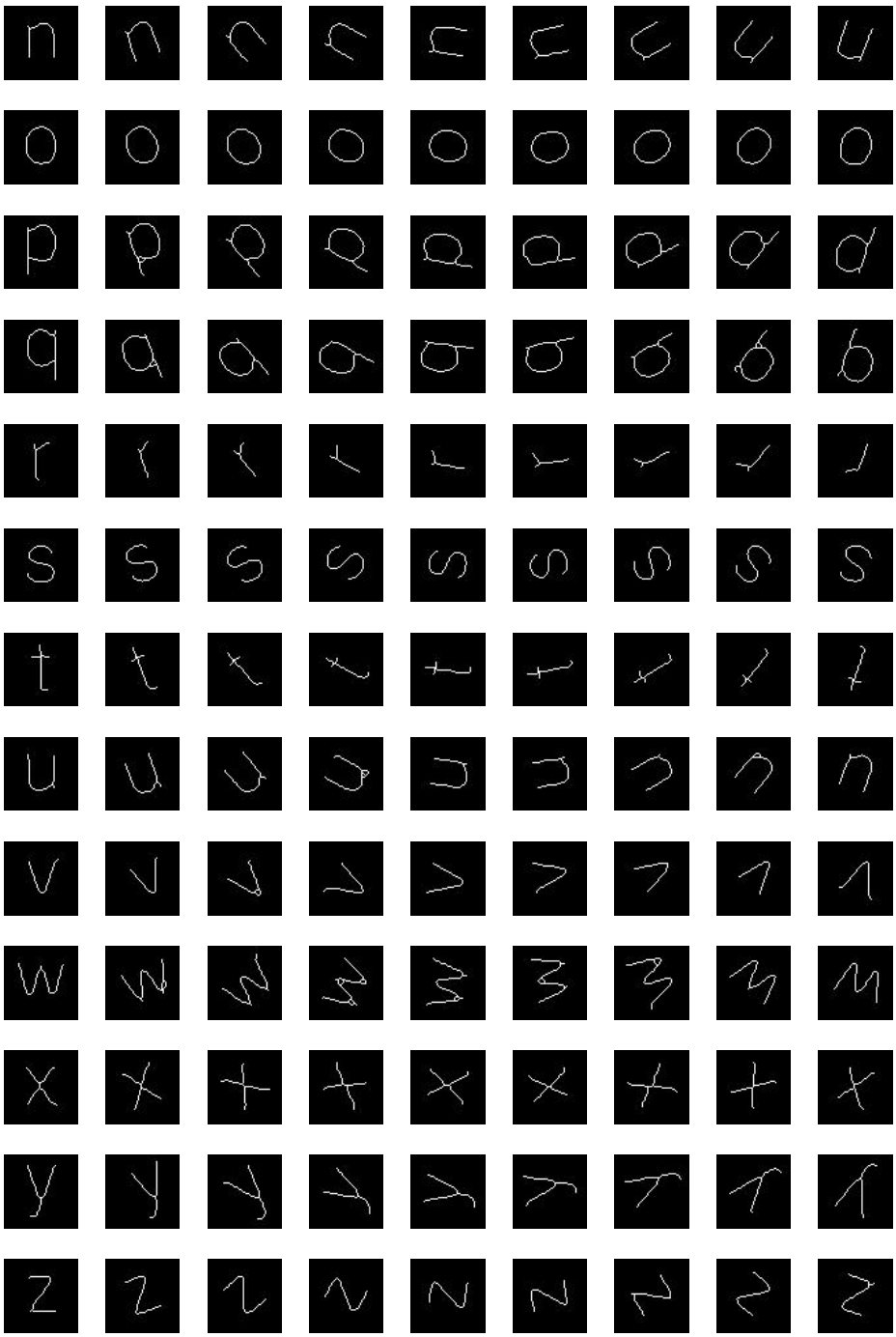


Figure 7.8: Training set of thinned, scanned characters (Part 2).

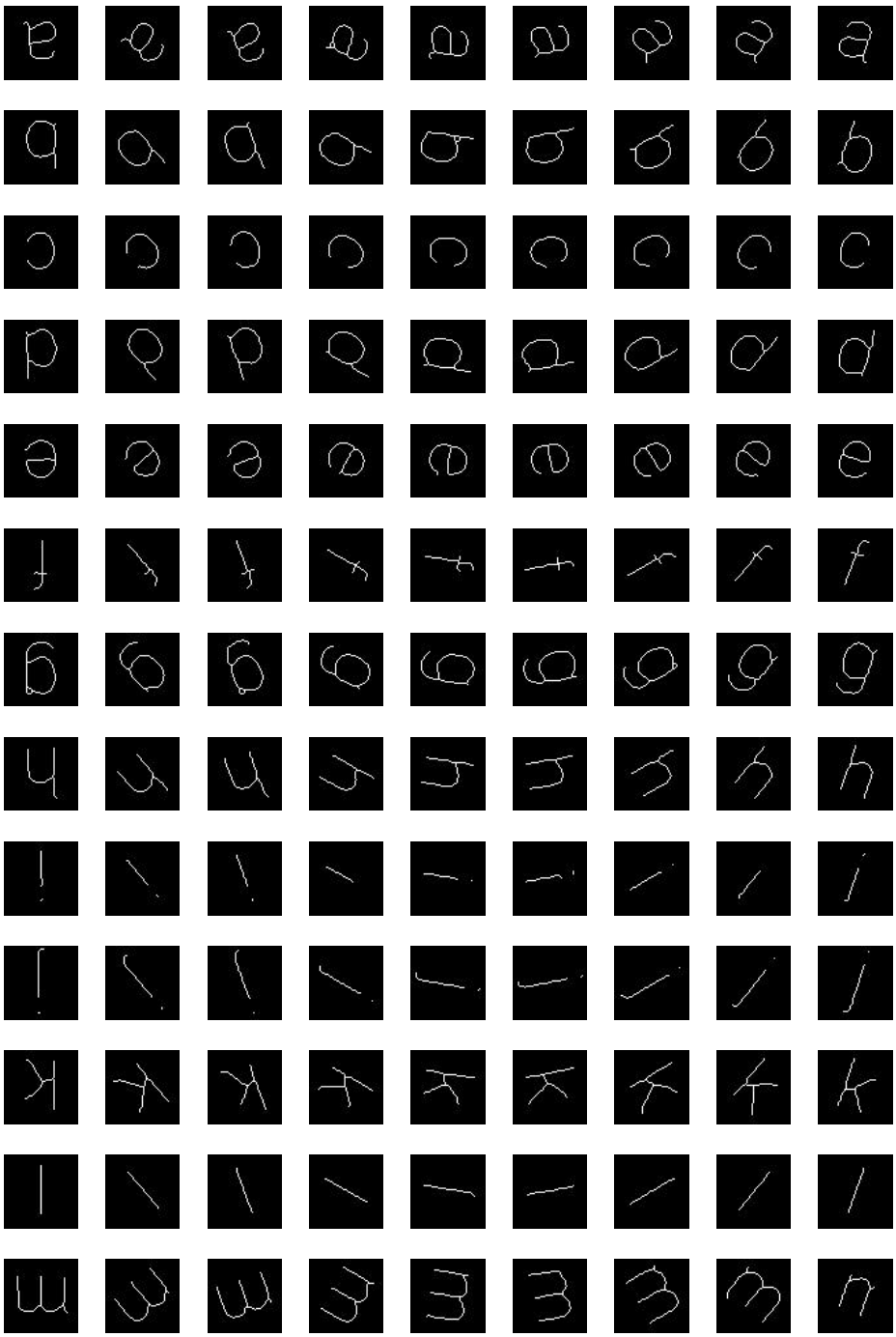


Figure 7.9: Test set of thinned, scanned characters (Part 1).

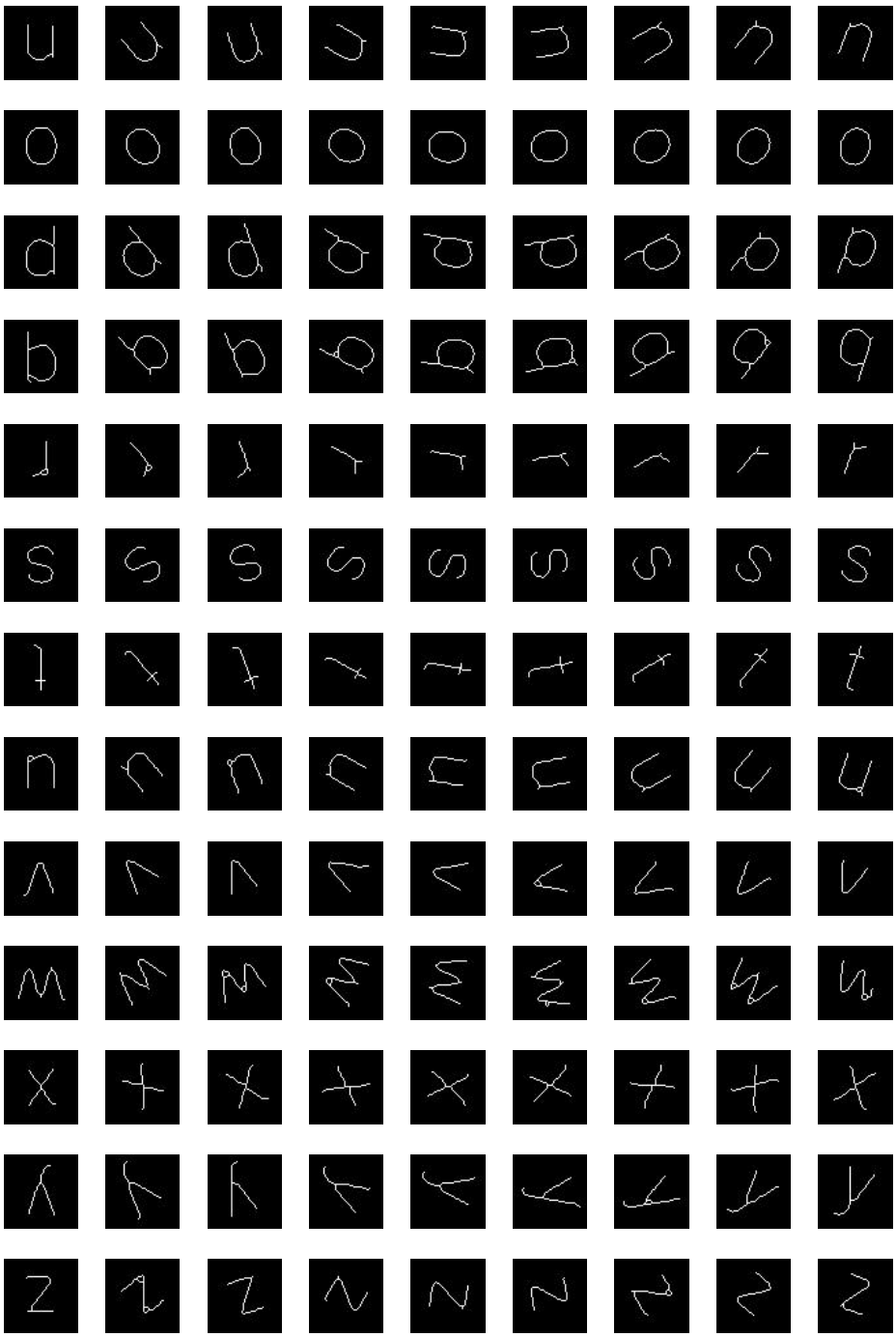
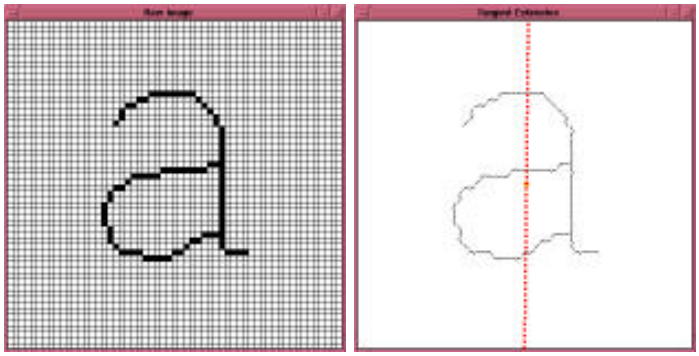
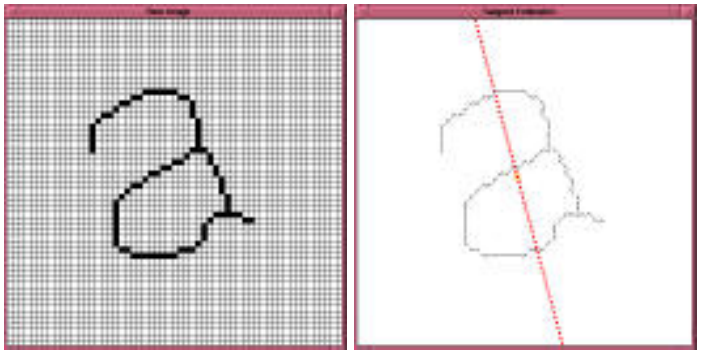


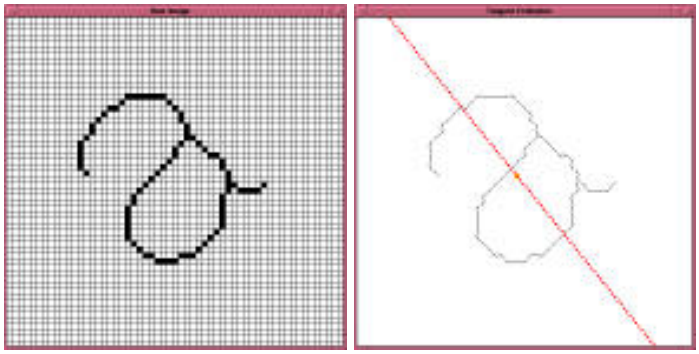
Figure 7.10: Test set of thinned, scanned characters (Part 2).



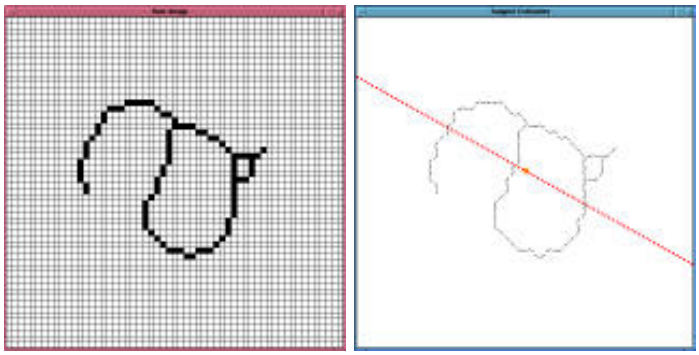
1. a00 Raw Image 2. a00 Tangents



3. a01 Raw Image 4. a01 Tangents

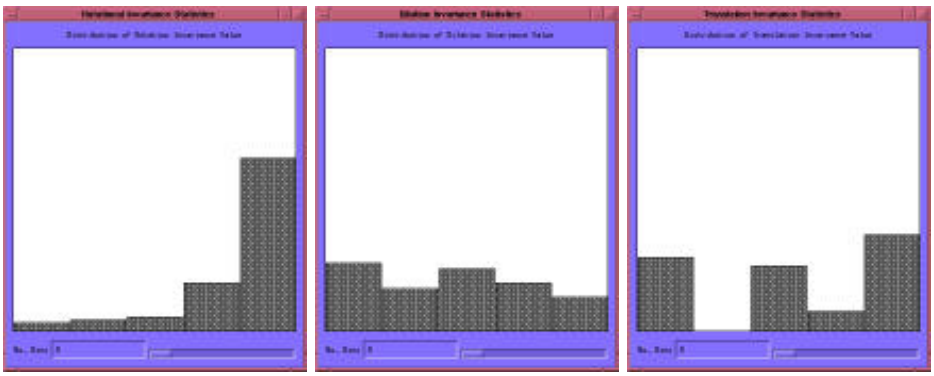


5. a02 Raw Image 6. a02 Tangents

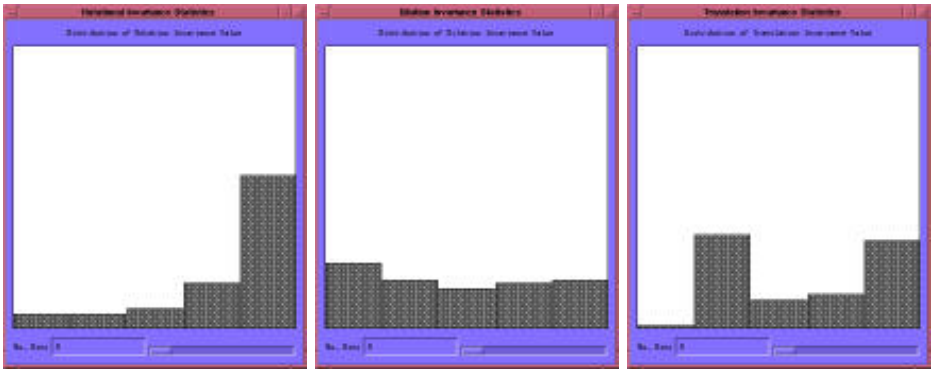


7. a03 Raw Image 8. a03 Tangents

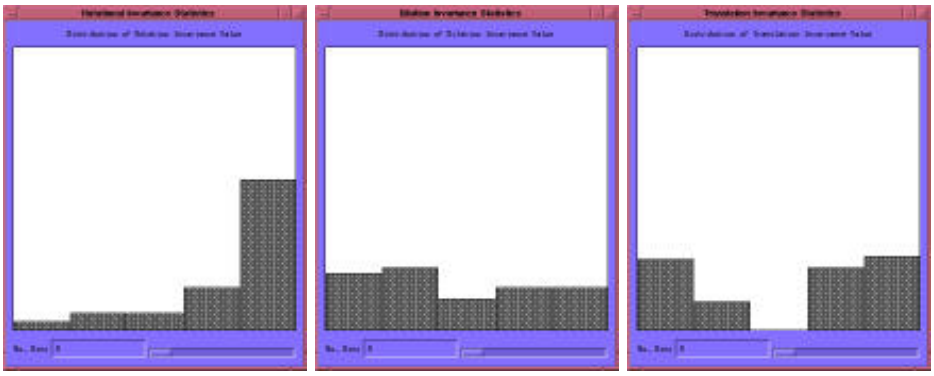
Figure 7.11: Tangents estimated for training examples of the letter "a".



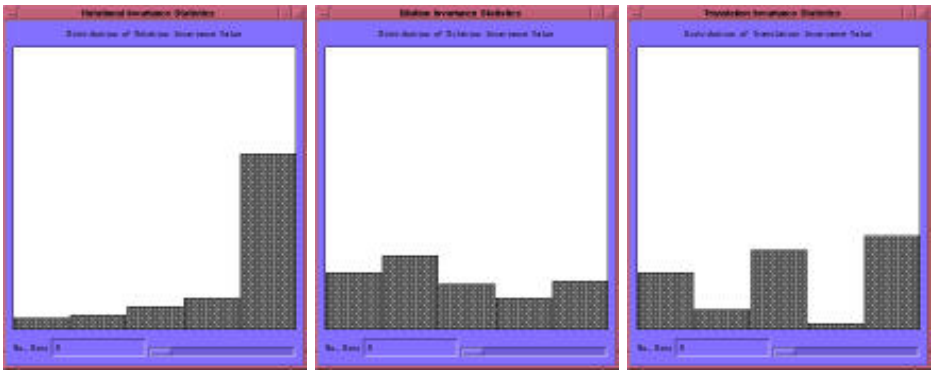
1. a00 Rotation 2. a00 Dilation 3. a00 Translation



4. a01 Rotation 5. a01 Dilation 6. a01 Translation

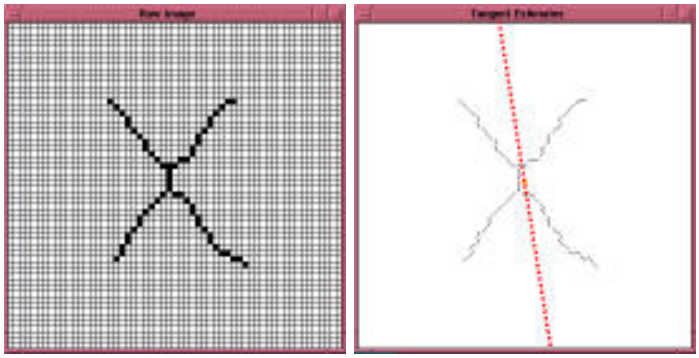


7. a02 Rotation 8. a02 Dilation 9. a02 Translation

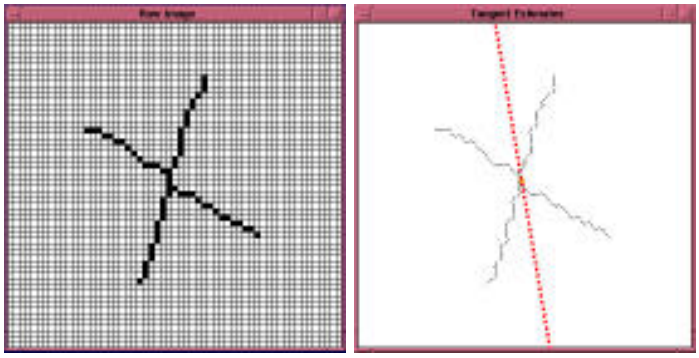


10. a03 Rotation 11. a03 Dilation 12. a03 Translation

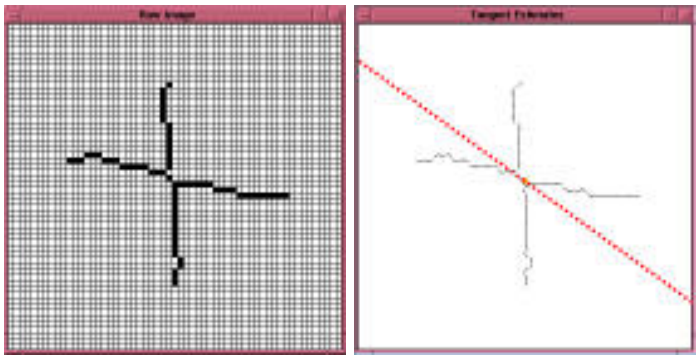
Figure 7.12: 5 bin Invariance Signatures for training examples of the letter "a".



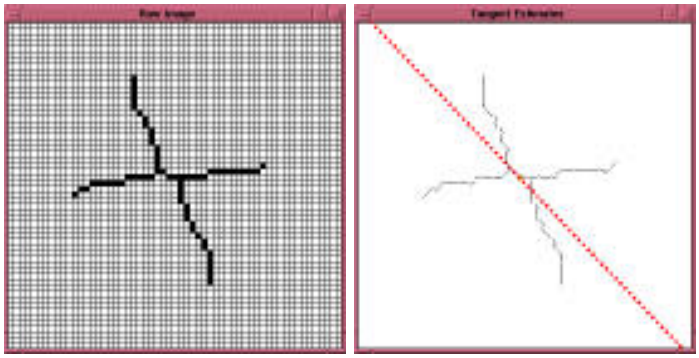
1. x00 Raw Image 2. x00 Tangents



3. x01 Raw Image 4. x01 Tangents

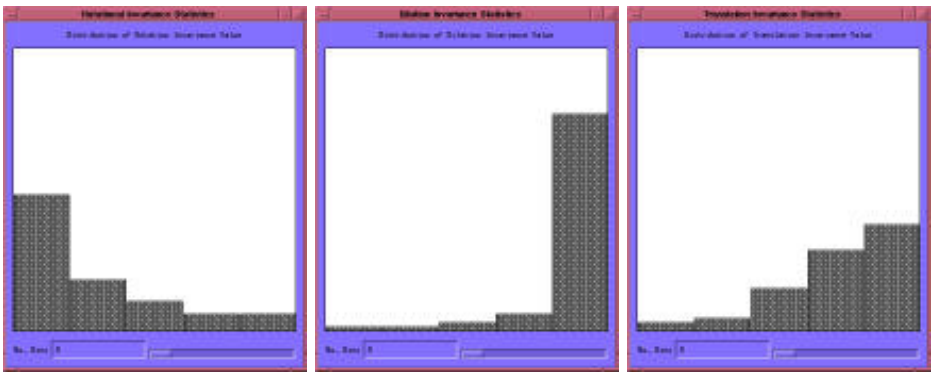


5. x02 Raw Image 6. x02 Tangents



7. x03 Raw Image 8. x03 Tangents

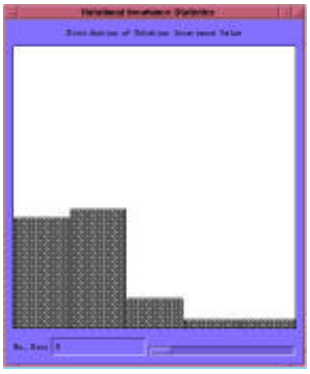
Figure 7.13: Tangents estimated for training examples of the letter "x".



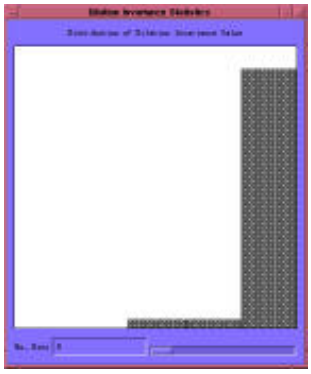
1. x00 Rotation

2. x00 Dilation

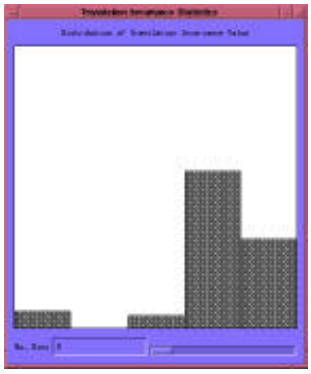
3. x00 Translation



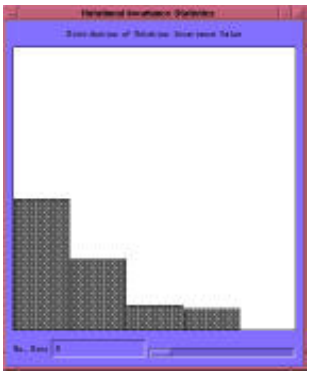
4. x01 Rotation



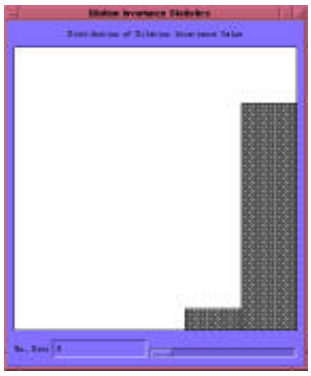
5. x01 Dilation



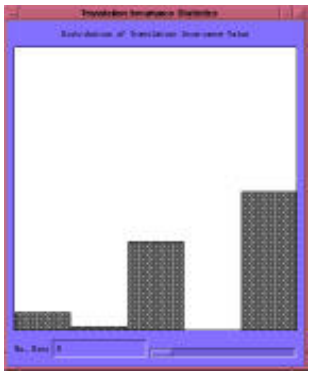
6. x01 Translation



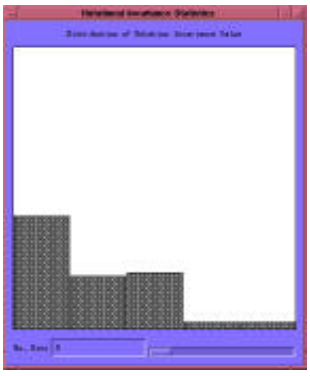
7. x02 Rotation



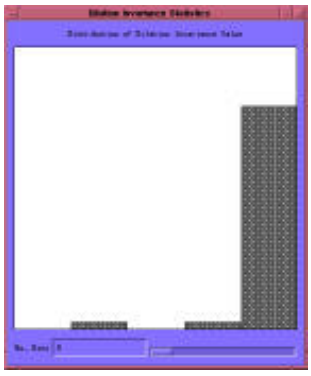
8. x02 Dilation



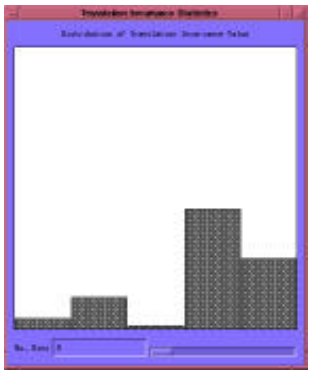
9. x02 Translation



10. x03 Rotation



11. x03 Dilation



12. x03 Translation

Figure 7.14: 5 bin Invariance Signatures for training examples of the letter “x”.

Chapter 8

Conclusion

The central tenet of this thesis is that the application of generic, problem-independent neural networks to invariant pattern recognition is both inefficient and inadequate. Such "black-box" neural networks have no inherent ability to perform invariant pattern recognition, and the only available information concerning their desired properties is that implicit in the data sets used to train them. This is a wasteful approach, since very large training sets, expensive in acquisition, storage and training time, are necessary to specify global invariance under transformations such as shift, rotation and scaling. This approach ignores the fact that the network designer usually knows a great deal about the desired performance of network, and can frequently express this knowledge in the form of a succinct, high-level rule. Something better is required.

Despite this, traditional neural networks have some very attractive properties. A network consists of a set of identical simple computational elements, joined by weighted connections. Computation is usually inherently parallel. Such characteristics make networks of this nature ideal candidates for implementation in hardware, such as specifically-fabricated VLSI circuits. With this in mind, we have sought means of constructing networks which retain these desirable properties, but are constrained to model the particular problem to which they are to be applied. We have called these Model-Based Neural Networks.

The Model-Based Neural Network approach may be thought of as a means of compiling an expert's high-level knowledge into the architecture of a neural network, and constraints on its weight-space. We have proposed a variety of different forms of Model-Based Neural Network in this thesis, but they share the characteristic that they consist of functionally-distinct modules. These modules perform component sub-tasks of the invariant pattern recognition problem, and their functionality is specified, either partially or completely, by the network designer. In this way, the designer can use expert *a priori* knowledge of the desired properties of the resultant network to constrain it so that it is guaranteed to have these properties. Invariance under transformations

such as shift, rotation and scaling, often desired in two-dimensional pattern recognition systems, can be made an *inherent* property of the network.

8.1 Using Weighting Functions to Constrain Networks

In Chapter 4 we introduced a form of Model-Based Neural Network in which, rather than being independent, the weights of the connections between nodes in a pair of layers are specified by a function of the nodes' coordinates and a vector of parameters. These parameters can be shared in various ways between sets of connections and nodes, allowing the dimensionality of the space which must be searched during training to be reduced. Moreover, the form of the weighting function can be used to constrain nodes and layers of nodes to implement specific classes of functions, such as spatial filtering, or integration of a filter's response. Once such a network is trained, knowledge of the weighting functions and parameter sharing is no longer required. All that remains is a set of nodes linked by weighted connections. The expert's knowledge, used to specify the architecture and constrain the weight-space during training, has been compiled into the network. Importantly, this allows properties to be introduced into the network completely independently of the training set.

A preliminary demonstration of the advantages of this approach was given by showing that it could be used to construct simple MBNNs which classified textures of differing absolute spatial frequencies correctly, independent of their orientations and positions. The MBNNs achieved 97 ± 7.0 percent correct classification of the two-class test set, compared with 53 ± 12 percent correct for the traditional neural networks. These networks were trained with textures of only one orientation, since rotation and scale invariance are inherent properties of their architectures. Moreover, these MBNNs were specified by only 18 parameters, compared with 908 parameters for the minimal TNNs for this problem. Weight Decay (see §3.3.1) failed to improve the performance of the TNNs, which is unsurprising considering the form of the training set. This simple example demonstrates the three main advantages of the MBNN approach: guaranteed invariant performance, greatly-reduced training set size and far fewer parameters. The last two properties result in much faster training.

In order to emphasize the improvements possible using this particular MBNN approach, a comparison was done with a study [Plaut and Hinton, 1987] which used a TNN to classify synthetic speech spectrograms as risers or non-risers (see §4.3). Expert knowledge indicated that this problem was in fact a case of the shift-invariant classification of signals containing differing line orientations, in the presence of noise. A variety of MBNNs was tried, and eventually a design was found, specified by only 22 parameters, which achieved 95.70 ± 6.25 percent correct classification of the test set after training with only 10 example patterns, using simulated annealing. Using a

1344 parameter TNN, Plaut and Hinton (1987) achieved 97.8 percent correct classification, after training with 250,000 example patterns using backpropagation. Again, the MBNN has achieved equivalent performance, but both the number of parameters and the training set size have been reduced by orders of magnitude.

8.2 The Invariance Signature Neural Network Classifier

In Chapter 5, a new invariant feature of two-dimensional contours was developed: the Invariance Signature. We believe that the Invariance Signature is a powerful descriptor of contour shape, which is closely-related to measures employed in human perception. Its application is by no means limited to Model-Based Neural Networks. It is a useful contribution in its own right.

Nevertheless, the development of the Invariance Signature was inspired by the desire to find an invariant contour descriptor which was suitable for calculation in a MBNN, and which corresponded well to theories of human contour perception. Since Lie group theory provides the link between the local changes in the positions of points under the action of a transformation and the global specification of the transformation, it provides the natural starting point. The Invariance Signature is a global measure of the degree of invariance of a given contour with respect to a set of Lie transformations, which, however, is constructed from *local* calculations. It is this that makes the Invariance Signature so attractive for use in a neural network.

The core of the Invariance Signature approach is this: rather than seeking individual invariant features of a contour, the Invariance Signature measures the *degree to which the contour is invariant* under a transformation. The statistics of these departures from invariance are themselves an invariant descriptor of the contour.

In Chapter 6, it was shown that a MBNN could be constructed which calculated a discrete version of the Invariance Signature, and used this as the basis for the classification of contours presented at the input layer. Since the Invariance Signature is shift-, rotation-, scale- and reflection-invariant, the output of this network is *guaranteed* to be invariant under the application of these transformations to the input layer. This network is called the Invariance Signature Neural Network Classifier. The ISNNC is a MBNN which consists of a wide variety of modules. These modules perform tasks as diverse as tangent vector estimation and the binning of data. The weights of all modules in the ISNNC except the final MLP classifier are specified independently of the training set; some are specified directly, others are determined by training the module on a sub-task. As with the weighting-function networks, the final result is just a collection of simple nodes joined by weighted connections, exactly as for a TNN.

In order to be useful, the Invariance Signature must not only be invariant, but it must retain sufficient information for contour classes to be distinguished. That this is

so is demonstrated in Chapter 7. The application chosen for the demonstration of the efficacy of ISNNCs was optical character recognition, the task being the classification of arbitrarily shifted and rotated lower-case alphabetic characters from a sans serif font. Two groups of experiments were performed.

The first group of experiments used noise-free, unambiguous computer-generated characters: perfect data. Networks were trained with only one upright example of each of the 22 unambiguous characters. The test set consisted of 4 differently shifted and rotated examples of each character. The best test test classification performance obtained by TNNs on these data was 15.00 ± 0.45 percent correct. Using NOEM (see §7.2.5), the ISNNC achieved 96.60 ± 0.00 percent correct classification, and with eigenvector-based local orientation extraction, 100.00 ± 0.00 percent correct: with perfect data, the ISNNC produces perfect results.

This preliminary experiment with synthetic "perfect" data demonstrated that the Invariance Signature was indeed sufficient to distinguish all the unambiguous letters of the alphabet. The second group of experiments used data obtained by scanning characters printed at 18 different orientations. Shifts were introduced by the segmentation process. These data were then thinned. The resulting data were far from perfect, the dominant problems being quantization noise and thinning artifacts. The data set was divided into a training set and a test set, each containing 9 examples of each character. Trained with this data, the TNNs achieved a best performance of 13.93 ± 0.30 percent correct. The ISNNCs obtained 72.74 ± 0.97 percent correct on ambiguous data, and 86.20 ± 1.18 percent correct on unambiguous data. Further inspection of the training and test data indicated other ambiguities. When corrected for these, ISNNC performance improved to 93.16 percent correct.

In all these cases, both for perfect data and for the scanned character task, the dimensionality of the classification module was very much lower for the ISNNCs than it was for the TNNs. For the scanned data task, the (minimal) TNN had 83018 parameters, whereas the ISNNC had only 881 parameters to be determined during training. Moreover, the dimensionality of the ISNNC classifier can be varied by the designer, by changing the number of bins for the discrete Invariance Signatures. The training time for these ISNNCs was correspondingly reduced.

These experiments indicate that the Invariance Signature can be successfully employed for the recognition of scanned characters independent of rotations and shifts, and that this technique can be implemented in a MBNN. The test set performance obtained is comparable to that obtained by others using thousands of training examples, and is remarkable considering the amount of noise present. Since the ISNNCs are guaranteed to be invariant under shift, scaling, rotation and reflection, and training set performance on the noisy data was 99.06 ± 0.63 percent correct, it can be concluded that test set performance below this level is due to failure to generalize in the presence

of noise, not failure to generalize in the sense of invariance. If the size of the training set were increased, test set performance could be expected to approach 100 percent correct.

8.3 Conclusion

Throughout this thesis, a number of different ways of designing Model-Based Neural Networks has been introduced. In every case, the MBNN was shown to out-perform a TNN on the same task. It might be said that the comparisons were unfair, since the training sets used were inadequate for good TNN performance to be expected. That only emphasizes one of the key advantages of the MBNN approach: large training sets are not required. Also, in every case the number of parameters required to specify the MBNN was smaller than that for the TNN, often dramatically so. The training time required was correspondingly less.

A new and powerful invariant descriptor for two-dimensional contours, the Invariance Signature, has been developed It has been demonstrated that the local nature of the initial calculations required to obtain the Invariance Signature makes it particularly suitable for implementation in a neural network. Networks based on the Invariance Signature have been designed, and successfully applied to the shift- and rotation-invariant recognition of scanned characters.

The Model-Based Neural Network approach to invariant pattern recognition has been successfully shown to provide a framework for a network designer to compile expert *a priori* knowledge of the problem domain into a neural network. These Model-Based Neural Networks have guaranteed invariances, require less training data, have fewer parameters, are faster to train, and out-perform their traditional counterparts.

Appendix A

Publications

Caelli, T. M., Squire, D. M. and Wild, T. P. (1993). Model-based neural networks, *Neural Networks* **6**: 613–625.

Squire, D. M. and Caelli, T. M. (1995). Shift, rotation and scale invariant signatures for two-dimensional contours, in a neural network architecture, *to appear in the Proceedings of the 1st International Conference on the Mathematics of Neural Networks and Applications (MANNA 95), Lady Margaret Hall, Oxford, published as a special edition of Annals of Mathematics and Artificial Intelligence*, J.C. Baltzer Scientific Publishing Company, Basel - Switzerland.

Appendix B

Introduction to the Xnet Neural Network Simulator

The **Xnet** Neural Network Simulator was written as a flexible environment in which to develop and test the Model-Based Neural Networks used in this thesis. It is written in C, for an X windows and Unix environment, using the Motif widget set. There is a graphical interface which allows the user to create networks of various architectures and connection paradigms interactively. Training and test sets can also be created using the program, but are more frequently generated by other systems.

Xnet provides powerful diagnostic output during the training of a network. Graphs of the training and test set errors and classification performances are updated after a user-specified number of iterations. The system can be made to "animate" the network's performance on a given input pattern as training proceeds, by assigning grey-levels to the nodes corresponding to their outputs after each iteration. These data allow the rapid detection of problems such as unit saturation, and allow the network architecture and training parameters for a given problem to be tuned quickly.

There is a "batch" version of the network training system which contains none of the overhead required for the monitoring of and interaction with the system required by the graphical user interface. This is optimized for fast training, and is used to train sets of networks once the architecture and training parameters have been decided upon using **Xnet**. This version still logs data during training, so that the dynamics of the training process may be investigated later, if necessary.

In all, the system consists of 11819 lines of code. It has been successfully compiled and used on both Silicon Graphics and Sun workstations. The code is available upon request via email to `squizz@cs.curtin.edu.au`.

The remainder of this appendix takes the form of three tutorials, which demonstrate the "look and feel" of the **Xnet** system, and the way it can be used to develop and analyse neural network solutions to problems. It is aimed at the novice user.

B.1 Familiarization with Xnet

B.1.1 Introduction

The aim of this tutorial is to become familiar with the use of the **Xnet** Neural Network Simulator. By the end, you should know how to load a neural network using **Xnet**, how to load sets of data for training neural networks with **Xnet**, and how to train networks. You will learn about the parameters used in training a feed-forward neural network using the backpropagation algorithm.

Next, you will learn how to construct your own networks using **Xnet**, and how to specify your own training sets. You will discover some of the capabilities of feed-forward networks trained with the backpropagation algorithm, and also some of their limitations.

B.1.2 Starting Xnet

Go to the directory where you have compiled the **Xnet** system, and simply type `Xnet` to launch the **Xnet** system. If the background colour of the **Xnet** window is blue when it starts up, it means that the **Xnet** resource database file has not been loaded for the display at which you are working. Typing `XN` in the directory where `Xnet` is located should fix this.

When **Xnet** has loaded, you should see the window shown in Figure B.1. As you can see, it is empty, except for a row of buttons on the right hand side of the window. In the course of this tutorial, you will learn what all of these buttons do.

B.1.3 Loading a Network

In order quickly to become familiar with the **Xnet** interface, it will be easiest to load a network that has been created earlier. To do this, click on the "Load Network" button. This will cause a dialog box to pop up, as shown in Figure B.2. To load a network file, you can click once on the filename, and then click "OK", or you can simply double-click on the filename. Load the network "ab.net".

When you have loaded the network, a diagram of its structure should appear in the left hand side of the **Xnet** main window. It should look like Figure B.3. In the **Xnet** program, networks are drawn with their input layer as the top layer of the network. The output layer is the bottom layer in the diagram. The network "ab.net", therefore, has a 15×15 input layer, a 3×3 hidden layer, and a 1×2 output layer.

B.1.4 Specifying an Input Vector For The Network

Now that you have a network loaded into the program, you need to specify an *input vector* so that you can run the network. The input vector specifies the outputs of

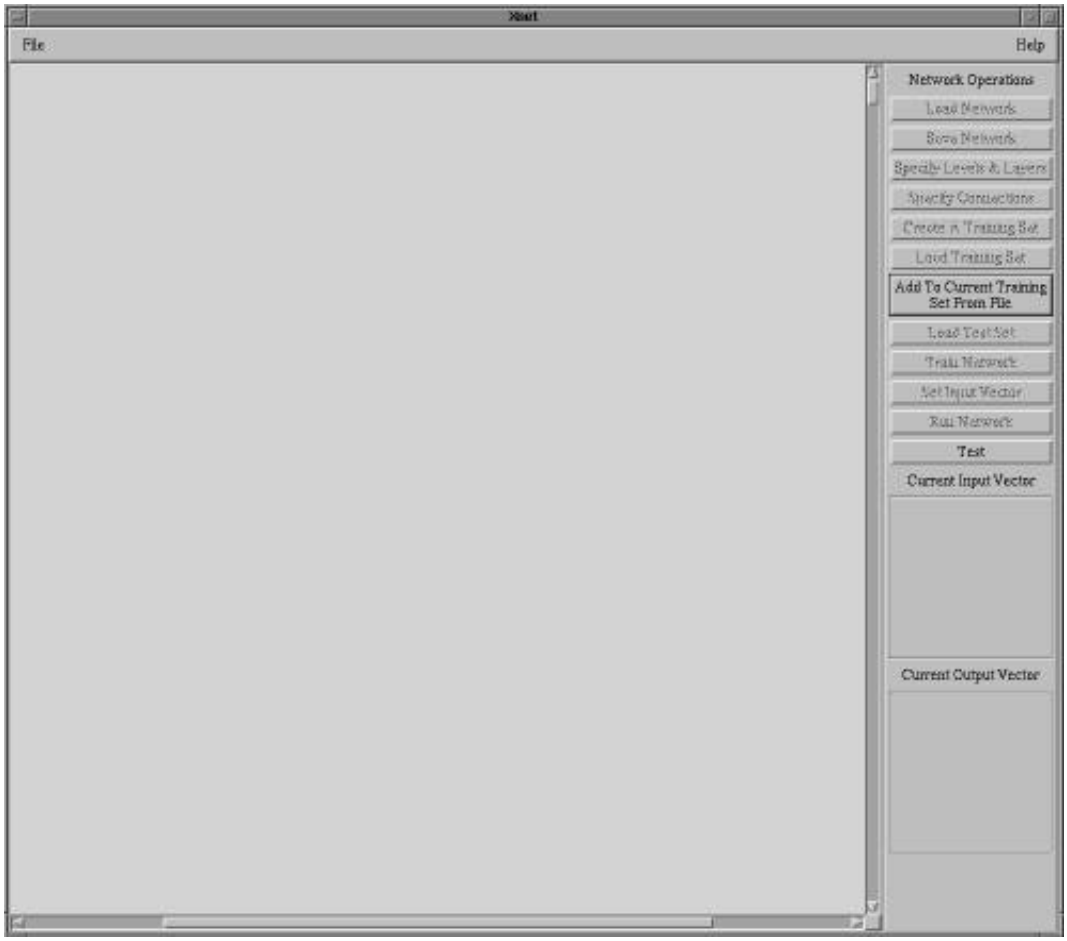


Figure B.1: **Xnet** main window on startup.

the nodes (artificial neurons) in the input layer of the network. These values are then propagated through the network to determine the output values of the nodes in the output layer.

To specify the input vector for the network, click on the button labeled "Set Input Vector". This will cause a window titled "Get Vector" to pop up, as shown in Figure B.4. This window contains tools. Most of the window (containing a hand-drawn "a" in Figure B.4) is a grid of squares of the same dimensions as the input layer of the network. Clicking in a square causes it to change to the current colour. You can draw in this area by holding down any mouse button in this area and dragging.

On the right hand side of the window is a column of rectangles of different grey levels. This is the palette from which you select the current colour, by clicking on the colour you want. Below this column is a square which displays the current colour. The various colours represent values in the vector in the range 0 to 1. Black represents 0 and white represents 1. Black and white are often the only values that you need. Along the bottom of the window are some buttons. Hopefully their functions are obvious from

Figure B.2: **Xnet** file load dialog box.

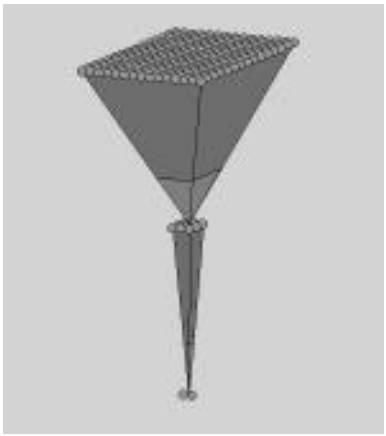


Figure B.3: **Xnet** network "ab.net".

their labels.

Practice drawing various different input vectors. When you have tried out the features available, click on "Clear to Zeros" to reset all the elements of the vector to zero. Then try drawing an "a" like the one in Figure B.4. When you are happy with your effort (it doesn't need to be a copy), click on the "Done" button.

Notice that the input vector you have just specified appears in the main **Xnet** window, below the column of buttons on the right. It should look like Figure B.5. Below the input vector, the current output vector is displayed. This is updated every time you modify the input vector, by running the input vector through the network and displaying the result. At the moment both the output nodes should have output values close to 0.5, because the weights in the network are initialized to ensure that this is the case before training. They should both be displayed as a mid-grey colour.

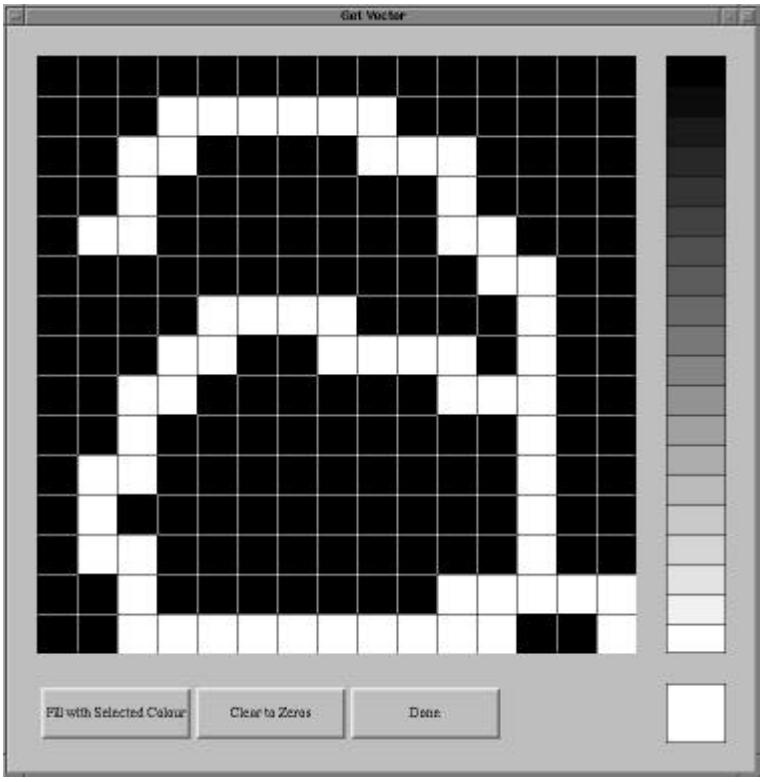


Figure B.4: Window for specifying a vector in **Xnet**.

B.1.5 Training Sets

In order for a neural network to be useful, values must be found for the connection weights so that the output of the network is correct (or approximately so) for a specified set of input vectors. This is usually achieved by presenting the network with a set of input vectors and the desired corresponding output vectors. This set of vectors is called a *training set*. The weights are then modified to minimize some cost function which indicates how well the network is performing at reproducing the behaviour specified by the training set.

In **Xnet**, training sets consist of a number of *training patterns*. Each training pattern consists of an output vector, and a set of input vectors. This format is used because it is very common to want many input vectors to produce the same output vector, especially in classification problems.

Loading A Training Set

A training set has been created to allow you to see how the format works. To load this training set into the program, click on the "Load Training Set" button in the main window. This will cause the dialog box in Figure B.2 to pop up. This time, you want to load the training pattern file "ab.pat".

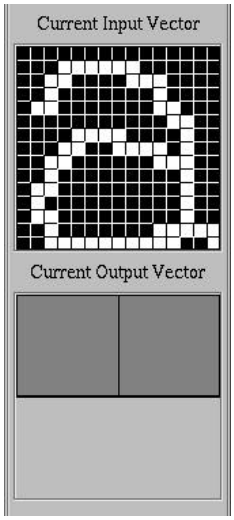


Figure B.5: Input and output vectors as displayed in the **Xnet** main window.

Viewing A Training Set

Note that the is a button in the main window labeled "Create, View or Edit A Training Set". After you have loaded "ab.pat", click on this button so that you can view the training set you have loaded. A window titled "Specify Training Set" will pop up, as shown in Figure B.6.

As you can see, Training Pattern 1 consists of an output vector for which the left node is 1 and the right node is 0, and nine input vectors, all of which are examples of the letter "a". There is another training pattern too. To see this, click in the darker grey area at the right of the button labeled "View/Edit Training Pattern Number:". The background should change to white, and a cursor will appear. Type "2", since you wish to view Training Pattern 2. Type "Enter" to cause this action to take place. A training pattern consisting of "b"s should appear. Notice that the upper group of buttons in the "Specify Training Set" window becomes greyed-out when you do this, and the lower group become active. At this stage you do not wish to modify the training set, so click on "Training Pattern Done", and then click on "Done" to return control to the main window.

B.1.6 The Training Control And Monitoring Window

Now that you have specified a network and a training set, you are in a position to train the network. To do this, click on the button labeled "Train Network" in the main window. This will cause a window labeled "Train Network" to pop up, as shown in Figure B.7.

This window allows you to set the parameters for the backpropagation algorithm and to monitor the progress of training. In the top left of the window are seven

Figure B.6: Window for creating, viewing or editing a training set.

fields where you can enter numbers that control training, and the display of training performance.

The Weight Update Equation

To understand these numbers, you need to be familiar with the weight update equation used in the backpropagation training algorithm. It is given in equation B.1.

$$w_{ij\,t+1} = w_{ij\,t} - \epsilon \left.\frac{\partial E}{\partial w_{ij}}\right|_t + \alpha \Delta_{t-1} - \lambda w_{ij\,t} \tag{B.1}$$

The Δ refers to the *total* change in w_{ij} at the previous iteration. Therefore,

$$\Delta_t = -\epsilon \left.\frac{\partial E}{\partial w_{ij}}\right|_t + \alpha \Delta_{t-1} - \lambda w_{ij\,t} \tag{B.2}$$

The symbols in equation B.1 correspond to the parameters in Table B.1

These parameters are all initialized to values that should work, though not necessarily optimally, in most situations. Do not change the Weight Decay Rate λ from zero, since it is not intended for use in this tutorial.

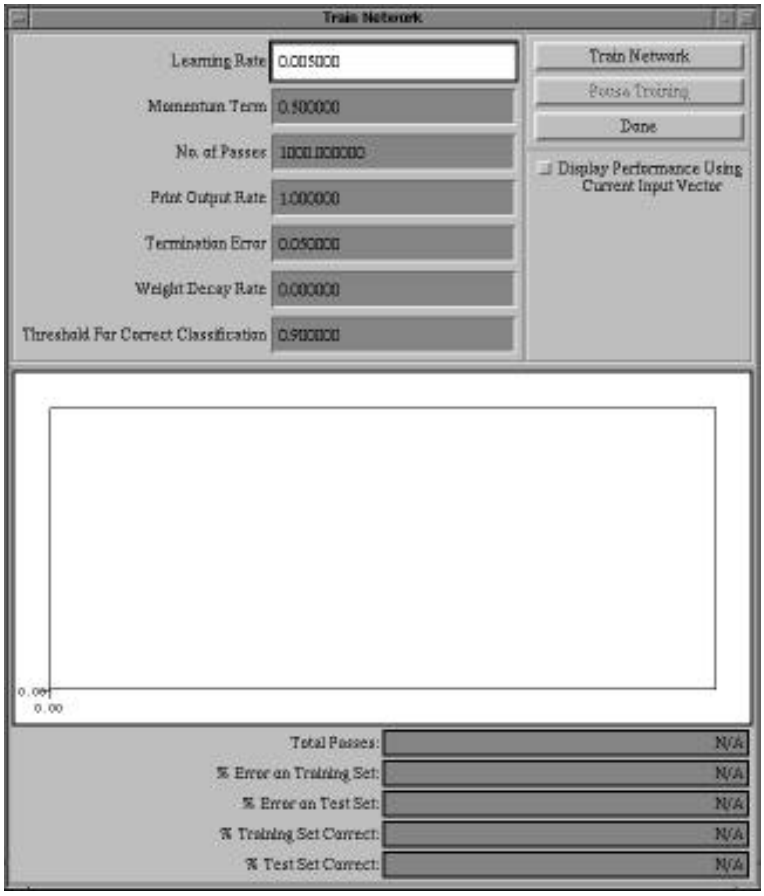


Figure B.7: Window for training a network.

Other Parameters

The other parameters which you can change control how long training goes on for, and how the progress is reported to you.

No. of Passes, as you would expect, controls the maximum number of passes over the training set that are to be made during training. The changes required in all the weights are accumulated for each input-output pair in the training set, and then the weights are updated at the end of each pass. A pass over the entire training set is often called an *epoch*.

Print Output Rate controls how often you are informed of the progress of training. Graphs and text values will be updated after every n passes, where n is the value typed in for **Print Output Rate**. You should not need to change it from 1 in this tutorial.

Termination Error provides a way to specify a condition whereby training will stop before the specified number of passes have been made. If the Sum Squared Error on the entire training set falls below the value specified here, training will stop. This value is given as a percentage, and allows you to specify a condition that is "good

Symbol	Parameter
$w_{ij\,t}$	Weight between nodes i and j at iteration t
$\left.\frac{\partial E}{\partial w_{ij}}\right\|_t$	Partial derivative of the error function E with respect to weight w_{ij} at iteration t
ϵ	Learning Rate
α	Momentum Term
λ	Weight Decay Rate

Table B.1: Training parameters

enough".

Threshold For Correct Classification determines when a given input vector will be deemed to have been classified correctly. It assumes that, for classification problems, there is one node in the output layer for every class. Each node is to have the value 1 for the class it corresponds to, and 0 for all other classes. There should, therefore, be only one node "on" for any given input pattern. Networks can only approximate this behaviour, the approximation improving as training progresses. The value here determines what is considered correct classification. It is initialized to 0.9. This means that the node which is supposed to be "on" must have an output value greater than 0.9, and all the others must have outputs less than (1 - 0.9) = 0.1 in order for the pattern to be considered correctly classified. This is not the only possible means of deciding upon classification, but is the one which will be used in this tutorial.

The "Display Performance Using Current Input Vector" Button

In the upper right of the "Train Network" window is a toggle button labeled "Display Performance Using Current Input Vector". When this toggle is on, it appears red. When on, this causes the input vector specified from the main window to be run through the network after each training pass, and the result displayed in the "Current Output Vector" region of the main window. Check this box, as watching this can be very useful in answering some of the questions below.

Training A Network

The initial values of all the parameters are suitable for the network ("ab.net") and training pattern ("ab.pat") which you have loaded. Click on the "Train Network" button to see what happens during the training of a network.

You should see that the axes on the graph in the middle of the window (see figure B.7) are labeled to reflect the desired number of passes. There will be up to four graphs plotted on these axes during training. The legend for their colours is given in Table B.2.

Graph Colour	Quantity Plotted
red	% Error on Training Set
black	% Error on Test Set
blue	% of Training Set Correctly Classified
green	% of Test Set Correctly Classified

Table B.2: Graph colours

The numerical values of these quantities after the most recent pass are displayed below the graph.

Now click "Done" to get back to the main window. Load the network "ab.net" from the file again, so that you can get it back to the state it was in before training. Change the control parameters to the values given in Table B.1.6.

Parameter	Value
Learning Rate	4
Momentum Term	0.5
No. of Passes	20
Print Output Rate	1
Termination Error	0
Weight Decay Rate	0

Table B.3: New values

Exercises

> **Aside:** These exercises are designed to help the user to understand the ways in which the **Learning Rate** and **Momentum** parameters influence the dynamics of training. Specifically the values have been chosen to demonstrate how the **Momentum** parameter can control oscillations, and how large values of the **Learning Rate** can lead to saturation.

Train the network with these parameters. Describe what you observe. Why do you think that this has happened?

Reload "ab.net". Now increase the **Momentum** to 0.7. Train the network with this new parameter. Describe what happened and why.

Reload 'ab.net". Change the **Momentum** back to 0.5, and increase the **Learning Rate** to 5. Train the network. Describe what happened and why.

Reload 'ab.net". Change the **Learning Rate** to 4.2. Train the network. Describe what happened and why.

B.2 Creating Networks And Training Sets with **Xnet**

B.2.1 Introduction

The aim of this tutorial is to build upon the work done in tutorial B.1, in order to complete your familiarization with **Xnet**. In doing this, you will study one of the classic problems from the history of neural networks, the XOR problem. This will give you some insights into the fundamental limitations of the simplest neural networks, and how to get around them.

B.2.2 The XOR problem

The first researchers in the field of neural networks were inspired by two main motivations. It had been discovered that brain cells, neurons, seemed to operated "summing" a collection of inputs, and then firing if a threshold was exceeded. The output of such a neuron would contribute to the input of many other neurons. Some researchers were interested in discovering how the brain might accomplish rational thought using such simple elements. Other researchers were interested in building devices (for tasks such as pattern recognition) based on such simple units – if it was good enough for the brain, why not for machines?

One of the first tasks researchers undertook was to demonstrate that networks of such simple units could perform all the basic logic functions (AND, OR, NOT *etc.*). One of the most interesting turned out to be the Exclusive OR function, or XOR. The exclusive OR of two logical variables is true if either of the variables, but not both, is true. It is false otherwise. The truth table is shown in Table B.4.

	0	1
0	0	1
1	1	0

Table B.4: XOR truth table.

This problem is easily mapped onto a neural network. The network must have an input layer with two nodes, and an output layer with one node. The values of these nodes should correspond to the values in the truth table.

B.2.3 Creating A Network

Specifying The Number Of Layers And Their Dimensions

Start the **Xnet** simulator, as described in tutorial B.1. Once **Xnet** has started, click on the button labeled "Specify Levels and Layers". This will cause a window titled

"Specify Network Dimensions" to pop up (see Figure B.8).

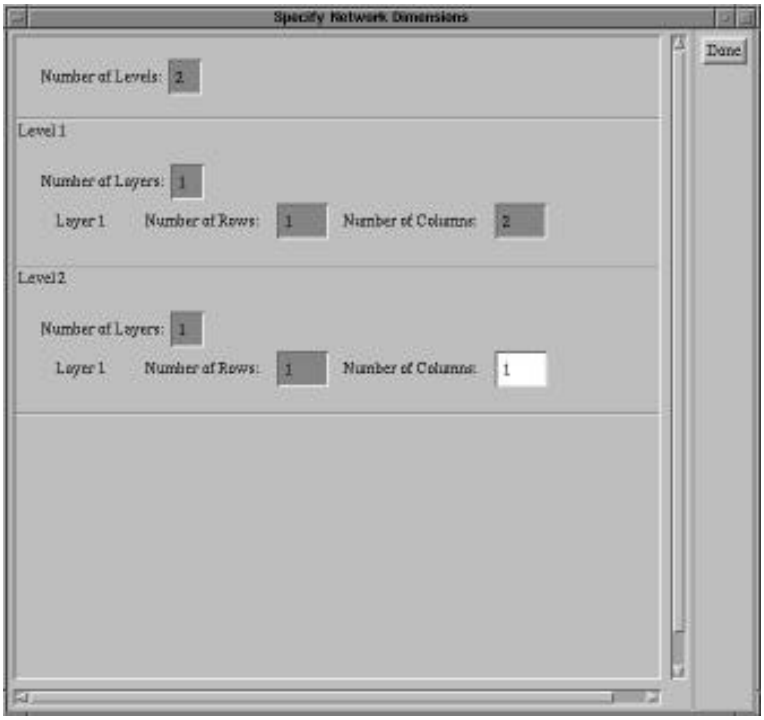


Figure B.8: **Xnet** window for specifying network dimensions.

Initially, only the topmost prompt, "Number of Levels", will be visible.[1] The other prompts will appear in response to the values you type in.

When designing a neural network to solve a problem, it is usually best to start with the simplest possible architecture. The simplest architecture for an attempt to solve the XOR problem has a 2 node input layer, a 1 node output layer, and nothing else. To construct this network, enter the numbers at each prompt as shown in Figure B.8. You need two levels, with one layer at each level. Level 1 is the input level, and the input layer should have one row of two columns. The last level, Level 2, is the output level. The output layer should have one row and one column.

You can navigate around the window by clicking in the dark grey data entry area you wish to edit. You can also move to the next data entry area by hitting `<tab>`. You can go back to the previous one using `<Shift>+<Tab>`.

Note: The value you type into a data entry area for a number of levels or a number of layers will not have any effect unless you hit return after typing it. The effect of changing one of these values is immediately

[1] The **Xnet** program allows the user to create networks other than the standard backpropagation networks used in these tutorials. These networks can have more than one *layer* of nodes at each *level* of the network. For the purposes of these tutorials, you will only ever need one layer per level.

visible, because it affects the number of other data entry areas in the scrolled window.

Once you have specified all the required values, click on "Done" to return to the main window. You should see a diagram of the network structure you have just specified, as shown in Figure B.9.

Figure B.9: **Xnet** two layer XOR network (still unconnected).

If the network which you have just created does not look like Figure B.9, click on "Specify Levels and Layers" and try again.

Specifying The Connections

Once you have created a system of layers of nodes, you then need to specify how those nodes are to be connected. The most common connection model for feed-forward neural networks is to have the outputs of all the nodes in the input layer connected to the inputs of all the nodes in the hidden layer, with each connection having an independent weight, and then for all the outputs of all the nodes in the hidden layer to be connected to all the inputs of all the nodes in the output layer, again with each connection having an independent weight.

To bring up the window for specifying how the layers you have specified should be connected, click on the button labeled "Specify Connections". This will bring up a window titled "Xnet: Connect Layers", as shown in Figure B.10.

This window, like the main window, consists of two main areas:

- A large area on the left containing a diagram of the network.
- An area of buttons on the right. These are in three groups:

  1. Buttons for setting the *Connection Model* for the connections between two layers.
  2. Buttons for setting the *Connection Function* for the connections between two layers.
  3. Buttons for connecting or disconnecting layers, and for indicating that you have finished making connections.

Figure B.10: **Xnet** window for specifying the way layers should be connected.

In **Xnet**, whenever a layer is a *source* for another layer (the *destination* layer), all the outputs of the nodes in the *source* layer are connected to all the inputs of all the nodes in the *destination* layer.

Xnet allows the user to create *Model-Based Neural Networks*, in which each connection is associated with a function rather than just with a number (the weight). Functions can dependent on the relative positions of the source and destination nodes, and on several parameters. These parameters may be shared between all nodes in a destination layer, between all connections entering a node, or they may be independent for each connection (in which case it behaves exactly like a normal neural network). You will not need to use any of these features.

The default *Connection Model* is "Connection". This is indicated by the diamond-shaped button on the left of the coloured label appearing depressed, and being red. The default *Connection Function* is "Normal", which means that each connection simply has an independent weight associated with it. This is indicated in the same way. Ensure that the "Connection Model" and "Connection Function" buttons are both set to their default values.

The quickest and easiest way to connect up the layers to form a standard neural

network is simply to click on the button labeled "Connect All Layers". This will cause each layer to be connected to the layer immediately below it. The connections are indicated by a "wedge" which starts out the same size as the base of the source layer, and finishes at a point at the centre of the destination layer.

Connect the network by clicking on "Connect All Layers", to see what the connected network looks like. This is the way you will usually connect networks. Then disconnect the network by clicking on "Disconnect All Layers".

Note: The wedges which indicate connections between layers in **Xnet** are colour-coded. The colour of the main body of the wedge indicates the *Connection Function* of those connections, and the colour of the tip indicates the *Connection Model*. The colours used match the colours of the buttons on the right of the "Connect Layers" window.

You can also specify connections by specifying a source layer and a destination layer, and then connecting them. To specify a layer as the source layer, click on the layer with the **left** mouse button. It will change colour, from orange to mustard-yellow (lighter than the default node colour). A layer is selected as the destination layer by clicking on it with the **middle** mouse button. It will change colour, to brown (darker than the default colour). Once a source and a destination layer have been specified, they can be connected by clicking on either layer with the **right** mouse button. Select the top (input) layer as a source layer, and the bottom (output) layer as a destination layer. Connect these layers by clicking on one of them with the right mouse button. Now that you have connected up your network, click on the "Done" button.

You will now be back at the main window, which now shows a diagram of your connected network. You will need to use this (untrained) network several times in the course of this tutorial, so you should save it. Click on the "Save Network" button. You will be prompted for a file name. Save the network as "xor1.net".

B.2.4 Creating a Training Set

You now have a network ready to train. Before you can train it though, you need a training set with which to do so. Click on the button labeled "Create, View, or Edit A Training Set" to pop up the "Specify Training Set Window".

You now need to specify the input and output vectors that correspond to the XOR problem. Remember that an **Xnet** training set consists of a number of *training patterns*. Each training pattern consists of an output vector, and an arbitrary number of input vectors for which the network should have that output. For the XOR problem, you have only two possible output vectors, since there is only one output node, which can have a desired output value of 0 or 1.[2] There are four possible input vectors. All the

[2] In a classification problem such as this, it is usual to specify the desired values of all output nodes

required values are specified in Table B.4.

At this stage there are no patterns in the training set. The first pattern you will add will be that for which the single output node has an output of 1. The corresponding input vectors are $\begin{bmatrix} 0 & 1 \end{bmatrix}$ and $\begin{bmatrix} 1 & 0 \end{bmatrix}$.

Click on the "Add New Training Pattern To Training Set" button, and then click on the "Set Output Vector" button. This will cause the "Get Vector" window to pop up (as described in tutorial B.1). The grid in which you draw the output vector has, in this case, only one square, occupying the entire drawing area. You must set its value to 1. Select the colour white (corresponding to 1) from the palette. Click in the drawing area. The drawing area should change colour to white. Click on the "Done" button.

The output vector you have just specified will now appear in the output vector area at the top left of the window, below the label "Training Pattern No. 1". You now need to specify the two corresponding input vectors. This is done by clicking on the "Add Input Vector" button, and using the "Get Vector" window to draw the desired input vector. You will need to do this twice. Remember that black (at the very top of the palette) corresponds to 0, and white to 1. Add the two input vectors to Training Pattern No. 1. When you have done this, the "Specify Training Set" window should look like that shown in Figure B.11, on the left.

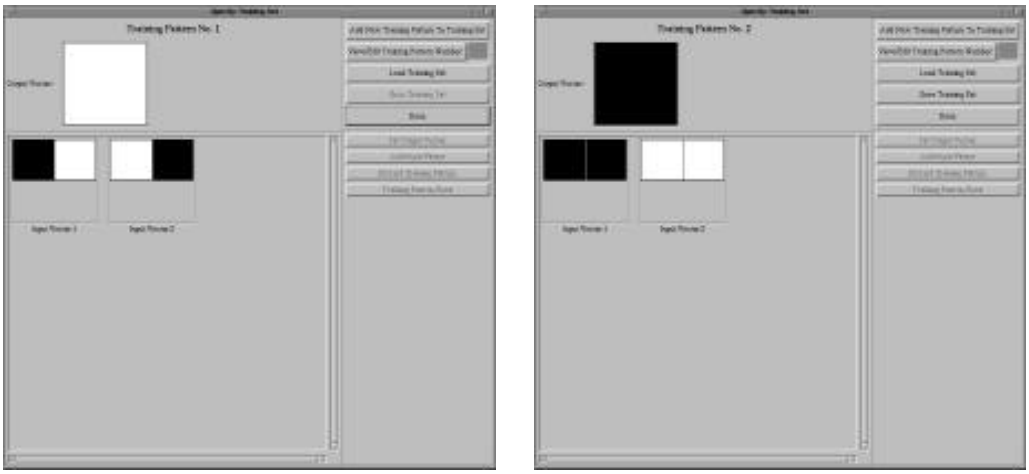


Figure B.11: Training patterns for the XOR problem.

as exactly 0 or 1. This is because there is usually a node corresponding to each class, and when an input vector of a given class is run through the network, the desired behaviour is for the output node corresponding to that class to be "on" (*i.e.* have an output of 1), and for all other nodes to be off. In some other sorts of neural network applications, such as engineering plant control or function estimation, this is not the case.

Click on the "Training Pattern Done" button to indicate that this training pattern is complete. This will return control to the upper group of buttons. Now you need to add a second training pattern to the training set. In this pattern, the desired value for the output node is 0, and the corresponding input vectors are $\begin{bmatrix} 0 & 0 \end{bmatrix}$ and $\begin{bmatrix} 1 & 1 \end{bmatrix}$. Add the required second training pattern to the training set, by repeating the procedure used to add the first pattern. When done, it should look like that shown on the right in Figure B.11.

You have now created a training set specifying the the XOR problem. You will need this training set later as well, so now you need to save it. Make sure that you have clicked on "Training Pattern Done" to return control to the upper set of buttons. Click on the "Save Training Set" button. You will be prompted for a file name. The default is "new.pat". Change this to "xor.pat", and then click on "OK". Then click 'Done" to return to the main window.

B.2.5 Training A Network To Solve The XOR Problem

The Two Layer Network

If you have followed the instructions provided, you now have in **Xnet** a two layer network that has not yet been trained, and a training set specifying the XOR problem. You will now attempt to train this network to solve the problem[3]. Click on "Train Network" to bring up the training window. Select whatever training parameters you think appropriate. Train the network, making no more than 10,000 passes over the training set. You can reload the saved version of the network, "xor1.net", to make another attempt at training. Make no more than three attempts.

No doubt you have discovered by now that the two-layer network fails to find a set of weights which solves the XOR problem. It is worth considering why, because it is to do with a fundamental property of the operation of any two layers of a neural network.

The Equations For Calculating The Output Of A Node In A Neural Network

The two-layer network used above is simple enough for us to be able to write down the equation describing its operation, and this will lead you to understand why it can't solve the XOR problem.

Every node in a traditional neural network (except those in the input layer) has a weighted input from the output of each of its source nodes, and one from the *bias*

[3]The various parameters used in training a neural network, and the graphs and other information that **Xnet** provides during the progress of training are described in tutorial B.1.

node.[4] The equation for the activation of node j, x_j, is thus:

$$x_j = \sum_{i=1}^{n} w_{ij} y_i + w_{bias} \tag{B.3}$$

where n is the number of source nodes for this node, y_i is the output of the ith node, and w_{ij} is the weight on the connection between nodes i and j. The activation, x_j, is used to calculate the actual output of the node by using it as the argument of a nonlinear function, here the usual sigmoid:

$$\begin{aligned} y_j &= f(x_j) \\ &= \frac{1}{1 + e^{-x_j}} \end{aligned} \tag{B.4}$$

The shape of the sigmoid is shown in Figure B.12.

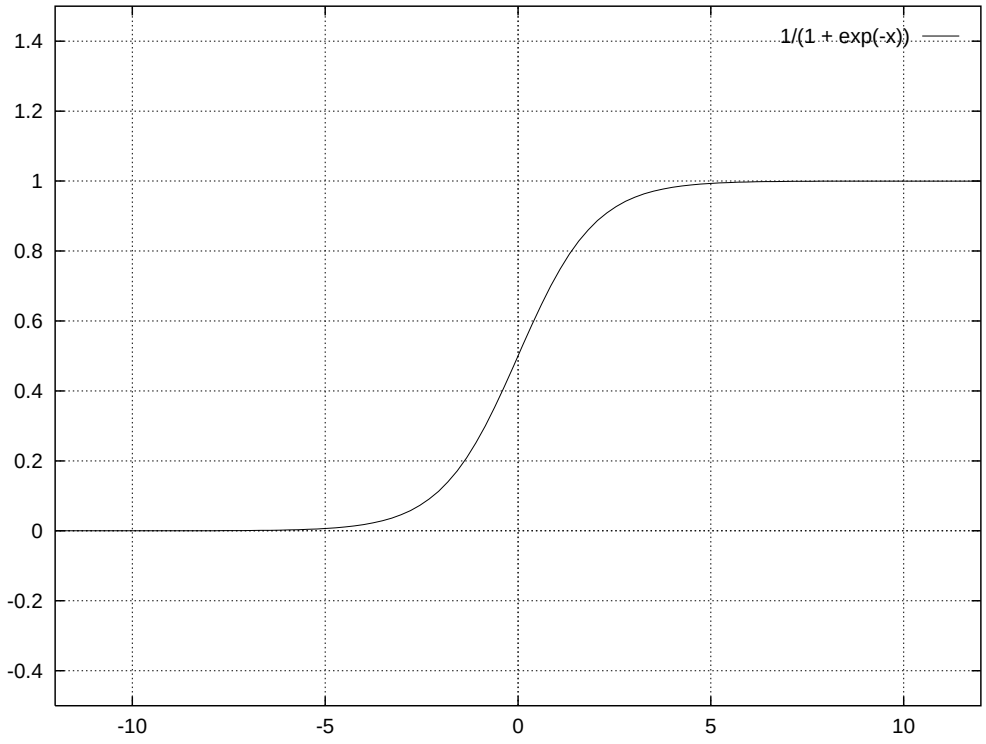


Figure B.12: The sigmoid function.

The equation for the output of the two-layer network used above can be written down in expanded form as shown in Equation B.5 (note that the index j has been

[4]Every node in a neural network for which the output is calculated (*i.e.* all except the input nodes) has a weighted connection to the *bias node*. This is a node which has a fixed output of 1. These thus provide an offset in the activation equation for each node. The negative of the weight on the connection to the bias node for a given node is analogous to a a threshold for that node.

dropped since there is only one output node).

$$x = w_1 y_1 + w_2 y_2 + w_{bias} \tag{B.5}$$

This is a very simple equation. Recall that for correct classification, we require that value of the output to be either above or below a threshold, which we will call θ. The fact that the output is "squeezed" by the sigmoid nonlinearity does not alter this, because the sigmoid is monotonic.

The variables y_1 and y_2 are constrained to only take on the values 0 or 1. For the network to solve the XOR problem, therefore, we require that the output be "on" (greater than the threshold) for two of the four possible combinations, and "off" for the the others. For successful performance then, we would need to find values of w_1, w_2 and w_{bias} so that the inequality in Equation B.6 is satisfied for inputs requiring a 1 output, and that in Equation B.7 for those requiring a 0 output. Note that the values of w_1, w_2, w_{bias} and θ must be the same in both equations.

$$w_1 y_1 + w_2 y_2 + w_{bias} > \theta \tag{B.6}$$

$$w_1 y_1 + w_2 y_2 + w_{bias} < \theta \tag{B.7}$$

Exercises

> **Aside:**This exercise is aimed at introducing the user to the notion of linearly separable problems.

Show that the two-layer network described by Equation B.5 can not solve the XOR problem. A graph using y_1 and y_2 as axes will be useful.

How does the above result apply to two-layer neural networks with more than two input nodes? Can you come up with a statement of the general class of problems that this kind of network can solve?

A Three-Layer Network We have seen that the XOR problem cannot be solved by a two layer network. We will now try a three layer network. Go to the **Xnet** main window. Click on the "Specify Levels & Layers" button to bring up the window allowing you to specify a new network. Create a network with:

- 3 levels, 1 layer per level
- An input layer with two nodes; 1 row, 2 columns
- A hidden layer with two nodes; 1 row, 2 columns
- An output layer with one node; 1 row, 1 column

Connect the network so that the input layer is a source for the hidden layer, and the hidden layer is a source for the output layer. Save the untrained network as "xor2.net".

Train the three-layer network using the training set created earlier. Record the parameters you used, and the performance obtained.

- Find the set of parameters that allow you to train the network in the least number of passes. Remember that you must reload the network from the file "xor2.net" before each attempt at training.
- Comment on the way the choice of parameters influences the % Error on Training Set (red graph) and the % Training Set Correct. It may be helpful to check the response of the trained network to various input patterns using the "Set Input Vector" button in the main window.

Why do you think that the three-layer network was able to solve this problem, whilst the two-layer network could not? What are the implications of this for larger networks?

There exists a network which can solve the XOR problem which has only four nodes. Can you find it? If so, describe it. (Hint **Xnet** allows you to connect a layer to *any* layer below it).

B.3 Generalization and Repeatability

B.3.1 Introduction

One of the claims often made about neural networks is that they have the ability to *generalize*. This means that they have the ability to classify correctly patterns which were not present in the training set. In order to do this, a neural network would have to somehow extract the "essence" of a class of patterns.

In this tutorial, you will investigate the ability of neural networks to generalize. You will see that neural networks certainly can generalize to some extent, but this capacity is limited. You will also investigate some training strategies aimed at improving generalization.

Another issue that is often overlooked in the study of neural networks is repeatability. When a neural network is created, the connection weights are initialized to random values. This means that the training procedure can proceed in differing ways for separate instances of the same network architecture even when using exactly the same training data. You will see how this can affect the final performance of networks after training.

B.3.2 Generalization

The claim that neural networks have the ability to *generalize* is one of the main reasons for their popularity. A pattern classification system that is able to classify correctly patterns which were not present in the training set used to produce it is clearly desirable. Such a system would be able to operate in the "real world", where input patterns might be noisy, or incomplete. A system based, for instance, on table lookup would fail in such an environment.

The Test Set

In order to evaluate the ability of a neural network to generalize, it is necessary to have a set of patterns other than the training set on which the network's performance can be tested. The **Xnet** simulator allows you to specify a *test set* for network. This is a set of patterns created in exactly the same way as those used for training the network. During training, if a test set is specified, the mean squared error and the classification performance of the network on the test set are evaluated after each pass over the training data. These results are displayed in the graph in the training window. In order to see how this works, we will do an example.

Start **Xnet**. Reload the network "ab.pat" that was used in tutorial B.1. Click on the "Specify Connections" button in the main window, and selected "OK" when warned about parameters being reinitialised. The "Connect Layers" window will pop up. Click on "Connect All Layers", and then click on "Done" to return to the main window. This procedure will ensure that you have an untrained network, with connection weights randomly initialized.

Now click on "Load Training Set", and load the pattern file "ab.pat". Click on "Create, View or Edit A Training Set". This will bring up the "Specify Training Set" window. Have a look at the basic style of the characters in each of the training patterns. To do this, type the number "1" in the grey box beside "View/Edit Training Pattern Number:", and then hit return. This will cause the lower set of buttons to become active. When you think you are familiar with the style of the characters, click on "Discard Training Pattern". This will cause the "a" pattern to be removed from the set. Then click on "'View/Edit Training Pattern Number:" again (since the "b" pattern is now pattern number 1). When you are familiar with the style of the "b"s, click on "Discard Training Pattern". There are now no patterns in the training set.

Now you must create a set of patterns to be used to evaluate the generalization performance of a network trained using the pattern file "ab.pat". You should be familiar with the procedure for creating a training set from tutorial B.2.

Create a set of patterns containing three examples of the letter "a", and three examples of "b". An example pattern is shown in Figure B.13. Do *not* copy this

example. Draw your letters freehand. Save these patterns in a file called "ab_test.pat".

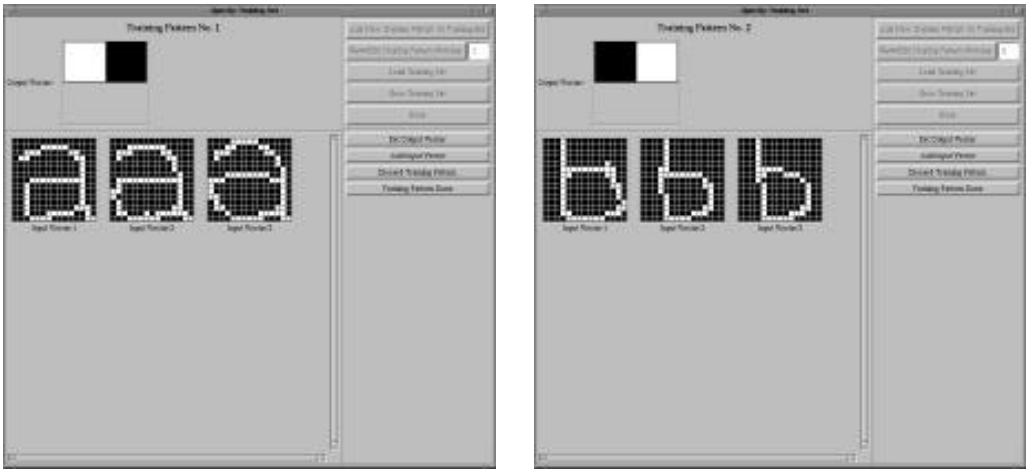


Figure B.13: Example test patterns for network trained using "ab.pat".

Then click "Done" to return to the main window.

Click "Load Training Set", and load the file "ab.pat" as the training set. Click "Load Test Set", and load the file you have just created, "ab_test.pat", as the test set for the network. Then train the network with the default training parameters.

Exercises

> **Aside:** These exercises are intended to introduce the user to the notion of over-training.

Record all the performance parameters of the network after 1000 passes over the training data. Describe the way the **% Error on Test Set** varied compared to the **% Error on Training Set**. What was the generalization performance of your network at the completion of training? Do you think that this performance is good? How might it be improved?

Train the network for another 1000 passes. Did the generalization improve with further training? If so, why do you think this was so?

B.3.3 Repeatability

The greater the number of patterns in the training and test sets, the longer it takes to train the network. Even on extremely fast computers it can take hours to train a

network if a realistically-sized training set is used. Since you may not have this amount of time available, some simulations have been done for you.

The task was to classify an input pattern as one of the digits from 0 to 9. The network architecture used is shown in Figure B.14.

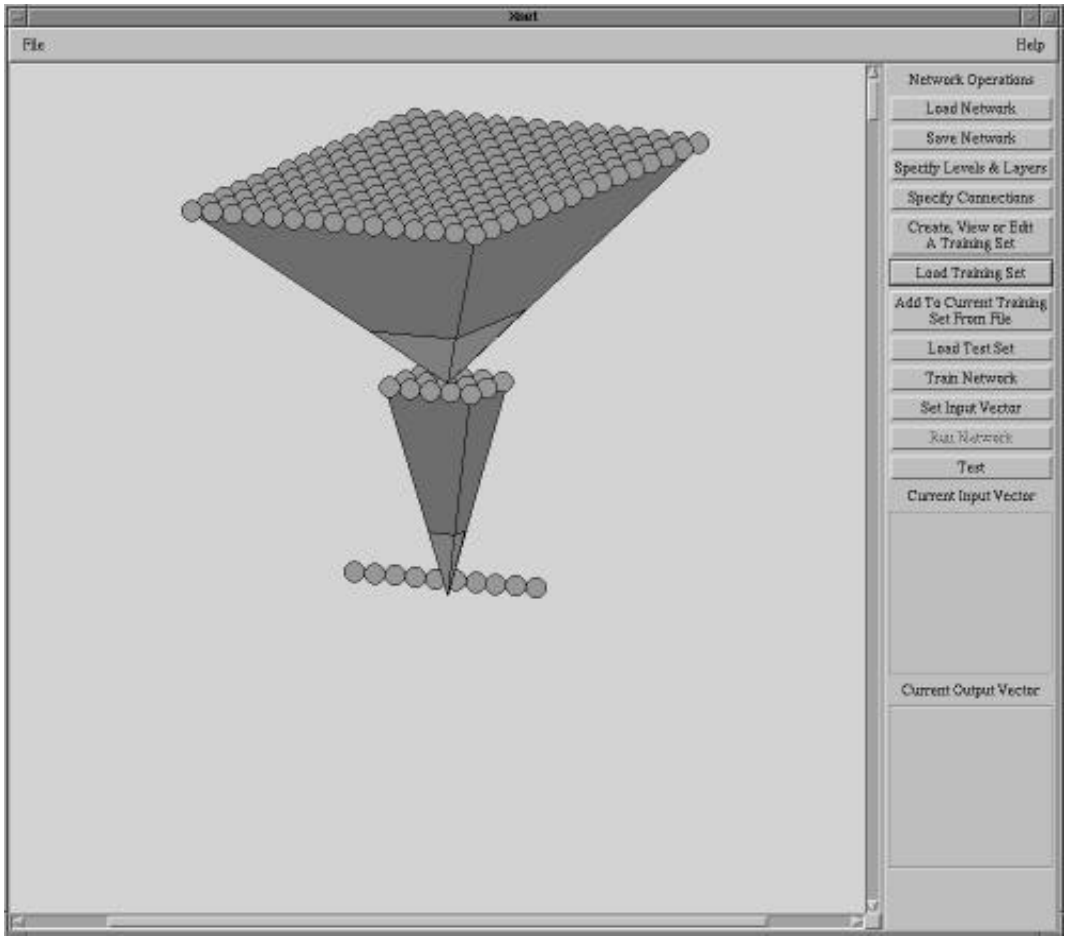


Figure B.14: Network used for classify patterns as a digit.

The training pattern consisted of 90 example patterns, 9 of each digit. Each input pattern had noise added to it. An example, for the digit "2", is shown in Figure B.15. These patterns can be found in the file "digits_train.pat". The test set also contained 90 patterns, each also with additive noise. These patterns can be found in the file "digits_test.pat".

Five networks of the architecture shown in Figure B.14 were created. Although they all had the same architecture, the connection weights of each network were randomly initialized. This meant that the initial conditions for training using the backpropagation algorithm were different for each network. Each network was trained for 4000 passes over the training data. Some of the graphs of network performance during training are shown in Figure B.16

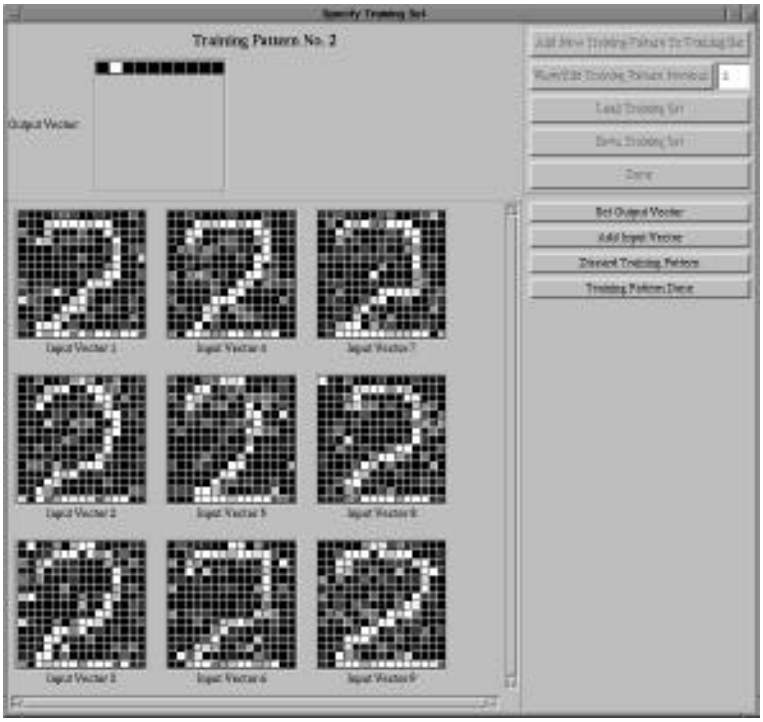


Figure B.15: Training pattern for the digit "2".

The final performances of the networks on the test and training data are summarized in Table B.5.

	Percent Correct		Percent Error	
	Training Set	Test Set	Training Set	Test Set
digits1.net	100	72.8	0.03	2.06
digits2.net	100	75.0	0.03	2.05
digits3.net	100	73.9	0.03	1.94
digits4.net	100	79.4	0.04	1.85
digits5.net	100	77.2	0.04	1.78
average	100	75.7	0.03	1.94

Table B.5: Final performance of networks trained on digit classification data.

Exercises

> **Aside:** These exercises are intended to encourage the user to think about the expected value of a networks classification performance, measures for quantifying generalization, and the importance of ensuring that results obtained from a randomly-initialized system are repeatable.

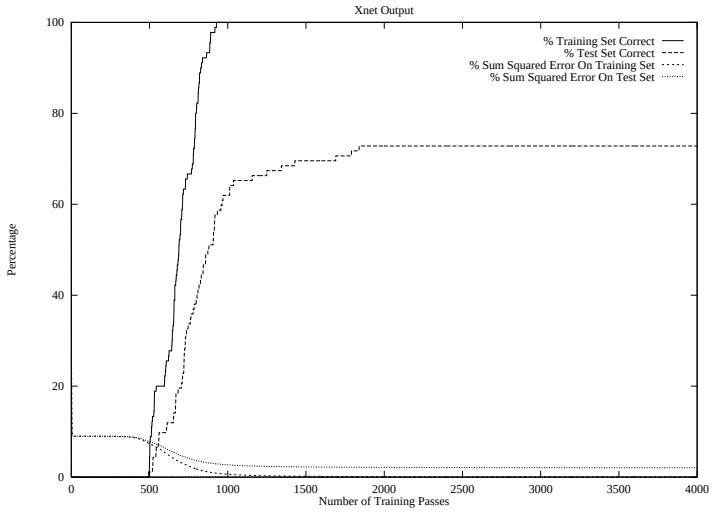
What would be the expected % Correct on the Test Set if the network was tested before

training, and was *forced* to classify each pattern (*i.e.* no "undecided" results allowed)?

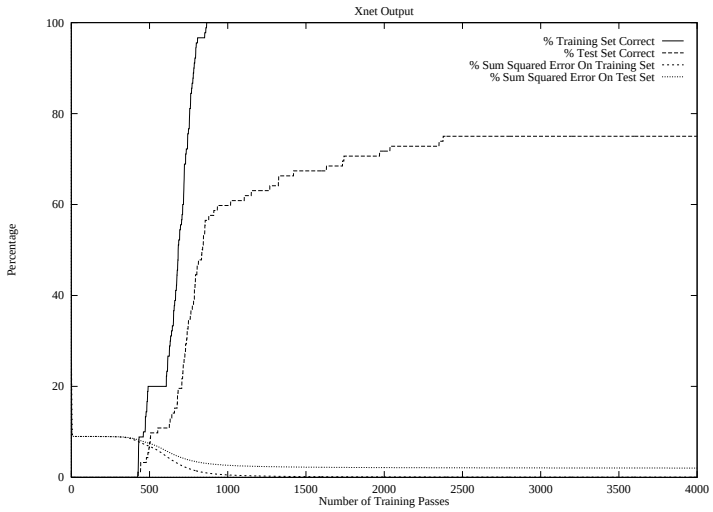
Do you consider the generalization performance of these networks to be good? What criterion could be used to judge the generalization performance of these networks?

It is clear that the performances of each of these networks after training with identical training sets and training parameters are different. Why is this so? Could this be avoided? What are the implications of this variability of performance for real-world applications of neural networks?

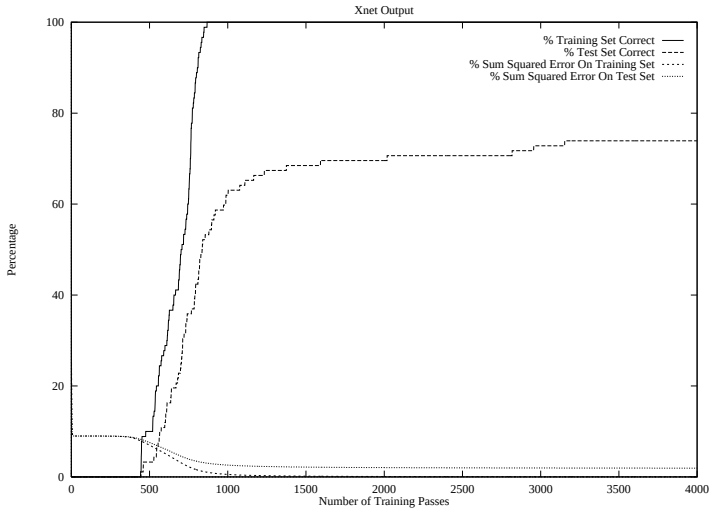
Finally, you might like to see how one of these pre-trained networks performs at recognizing your own hand-drawn digits. Load the network "digits1.net" into the **Xnet** simulator. You can qualitatively evaluate the performance of the network at classifying your own hand-drawn digits by clicking on "Set Input Vector" in the main window. This causes the "Get Vector" window to appear. Anything you draw in this window immediately gets run through the network, and the resultant output vector is displayed the the "Current Output Vector" section of the main window. Try to determine which features of each digit are important for correct classification. What was the performance of the network on your patterns? Do you consider this good? Can you suggest a more robust system for classifying hand-written characters?



(a) Network 1



(b) Network 2



(c) Network 3

Figure B.16: Performance of networks trained on digit classification data.

Bibliography

Aarts, E. and Korst, J. (1989). *Simulated annealing and Boltzmann machines*, John Wiley and Sons, New York.

Ade, F. (1983). Characterisation of textures by "eigenfilters", *Signal Processing* **5**: 451–457.

Ahmed, N. and Rao, K. (1975). *Orthogonal transforms for digital signal processing*, Springer-Verlag, New York.

Altmann, J. and Reitböck, H. J. (1984). A fast correlation method for scale- and translation-invariant pattern recognition, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **6**(1): 46–57.

Amari, S.-I. (1977). Neural theory of association and concept-formation, *Biological Cybernetics* **26**: 175–185.

Amari, S.-I. (1990). Mathematical foundations of neurocomputing, *Proceedings of the IEEE* **78**(9): 1443–1463.

Asriel U. Levin, T. K. L. and Moody, J. E. (1994). Fast pruning using principal components, *Advances in Neural Information Processing Systems* **6**: 35–42.

Austin, J. (1989a). An associative neural architecture for invariant pattern classification, *IEE First International Conference on Artificial Neural Networks*, Conf. Publ. No. 313, IEE, pp. 196–200.

Austin, J. (1989b). High speed invariant recognition using adaptive neural networks, *IEE 3rd International Conference on Image Processing and its Applications*, Conf. Publ. No. 307, IEE, pp. 28–32.

Ballard, D. H. and Brown, C. M. (1982). *Computer Vision*, Prentice–Hall, Englewood Cliffs, New Jersey 07632.

Barnard, E. and Botha, E. C. (1993). Back-propagation uses prior information efficiently, *IEEE Transactions on Neural Networks* **4**(5): 794–802.

Barnard, E. and Casasent, D. (1990). Shift invariance and the Neocognitron, *Neural Networks* **3**(4): 403–410.

Barrett, E., Gheen, G. and Payton, P. (1993). Representation of three-dimensional object structure as cross-ratios of determinants of stereo image points, *in* Mundy, Zisserman and Forsyth (1993), pp. 47–68.

Baum, E. B. and Haussler, D. (1989). What net size gives valid generalization?, *Neural Computation* **1**: 151–160.

Bishop, C. M. (1995). *Neural networks for pattern recognition*, Clarendon Press, Oxford.

Bulsari, A. (1993). Some analytical solutions to the general approximation problem for feedforward neural networks, *Neural Networks* **6**(7): 991–996.

Caelli, T. and Dodwell, P. (1984). Orientation-position coding and invariance characteristics of pattern discrimination, *Perception and Psychophysics* **36**(2): 159–168.

Caelli, T. M. and Liu, Z.-Q. (1988). On the minimum number of templates required for shift, rotation and size invariant pattern recognition, *Pattern Recognition* **21**(3): 205–216.

Caelli, T. M., Squire, D. M. and Wild, T. P. (1993). Model-based neural networks, *Neural Networks* **6**: 613–625.

Caelli, T., Preston, G. and Howell, E. (1978). Implications of spatial summation models for processes of contour perception: A geometric perspective, *Vision Research* **18**: 723–734.

Carpenter, G. A. and Grossberg, S. (1987). Invariant pattern recognition and recall by an attentive self-organizing ART architecture in a nonstationary world, *IEEE First International Conference on Neural Networks*, SOS Printing, San Diego, CA, USA., IEEE Service Center, pp. II/737–745.

Carpenter, G. A. and Grossberg, S. (1988). The ART of adaptive pattern recognition by a self-organizing neural network, *IEEE Computer* **21**(3): 77–90.

Casasent, D. and Psaltis, D. (1976). Scale invariant optical transform, *Optical Engineering* **15**: 258–261.

Chen, Y.-S. and Hsu, W.-H. (1988). A modified fast parallel algorithm for thinning digital patterns, *Pattern Recognition* **7**: 99–106.

Chow, G. and Li, X. (1993). Towards a system for automatic facial feature detection, *Pattern Recognition* .

Cole, J. B., Murase, H. and Naito, S. (1991). A Lie group theoretic approach to the invariance problem in feature extraction and object recognition, *Pattern Recognition Letters* **12**: 519–523.

Delopoulos, A., Tirakis, A. and Kollias, S. (1994). Invariant image classification using triple-correlation-based neural networks, *IEEE Transactions on Neural Networks* **5**(3): 392–408.

Fahlman, S. E. (1988). An empirical study of learning speed in back-propagation networks, *Technical Report CMU-CS-88-162*, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.

Fahlman, S. E. (1991). The recurrent Cascade-Correlation architecture, *Technical Report CMU-CS-91-100*, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.

Fahlman, S. E. and Lebiere, C. (1990). The Cascade-Correlation learning architecture, *Technical Report CMU-CS-90-100*, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Faugeras, O. (1993). Cartan's moving frame method and its application to the geometry and evolution of curves in the euclidean, affine and projective planes, *in* Mundy et al. (1993), pp. 11–46.

Ferraro, M. and Caelli, T. (1988). The relationship between integral transform invariances and Lie group theory, *Journal of the Optical Society of America (A)* **5**: 738–742.

Ferraro, M. and Caelli, T. M. (1994). Lie transform groups, integral transforms, and invariant pattern recognition, *Spatial Vision* **8**(1): 33–44.

Fontaine, T. and Shastri, L. (1992). Handprinted digit recognition using spatiotemporal connectionist models, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 169–175.

Forsyth, D., Mundy, J. L. and Zisserman, A. (1992). Transformational invariance - a primer, *Image and Vision Computing* **10**(1): 39–45.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, *Biological Cybernetics* **36**: 193–202.

Fukushima, K. (1975). Cognitron: A self-organizing multilayered neural network, *Biological Cybernetics* **20**: 121–136.

Fukushima, K., Miyake, S. and Ito, T. (1983). Neocognitron: A neural network model for a mechanism of visual pattern recognition, *IEEE Transactions on Systems, Man and Cybernetics* **13**(5): 826–834.

Gibson, J. (1966). *The Senses Considered as Perceptual Systems*, Houghton-Mifflin, Boston, Massachusetts.

Gros, P. (1993). How to use the cross ratio to compute projective invariants from two images, *in* Mundy et al. (1993), pp. 107–126.

Grossberg, S. (1980). How does a brain build a cognitive code?, *Psychological Review* **87**: 1–51.

Hebb, D. O. (1949). *The Organization of Behaviour*, Wiley, New York.

Himes, G. S. and Iñigo, R. M. (1992). Automatic target recognition using a Neocognitron, *IEEE Transactions on Knowledge and Data Engineering* **4**(2): 167–172.

Hoffman, W. C. (1966). The Lie algebra of visual perception, *Journal of Mathematical Psychology* **3**: 65–98.

Hoffman, W. C. (1978). The Lie transformation group approach to visual neuropsychology, *in* E. Leewenberg and H. Buffart (eds), *Formal theories of visual perception*, Wiley, New York, pp. 27–66.

Holden, S. B. and Anthony, M. (1992). Quantifying generalization in linearly weighted neural networks, *Technical Report LSE-MPS-42*, London School of Economics Mathematics.

Hu, M. K. (1962). Visual pattern recognition by moment invariants, *IEEE Transactions on Information Theory*, Vol. IT-8, pp. 179–187.

Hubel, D. and Wiesel, T. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex, *Journal of Physiology* **160**: 106–154.

Jain, A. K. (1989). *Fundamentals of digital image processing*, Prentice-Hall information and system sciences series, Prentice-Hall International, London.

Khotanzad, A. and Lu, J.-H. (1990). Classification of invariant image representations using a neural network, *IEEE Transactions on Acoustics, Speech, and Signal Processing* **38**(6): 1028–1038.

Kirkpatrick, S., Jr., C. G. and Vecchi, M. (1983). Optimization by simulated annealing, *Science* **220**(4598): 671–680.

Kohonen, T. (1972). Correlation matrix memories, *IEEE Transactions on Computers* **C-21**: 353–359.

Kree, R. and Zippelius, A. (1988). Recognition of topological features of graphs and images in neural networks, *Journal of Physics A: Mathematical and General* **21**: L813–L818.

Krogh, A. and Hertz, J. A. (1991). Dynamics of generalization in linear perceptrons, *Advances in Neural Information Processing Systems* **3**: 897–903.

Krogh, A. and Hertz, J. A. (1992). A simple weight decay can improve generalisation, *Advances in Neural Information Processing Systems* **4**: 950–957.

Lautrup, B., Hansen, L. K., Law, I., Mørch, N., Svarer, C. and Strother, S. (1994). Massive weight-sharing: A cure for extremely ill-posed problems, *in* H. Hermann, D. Wolf and E. Pöppel (eds), *Workshop on Supercomputing in Brain Research: From Tomography to Neural Networks, Jülich, Germany*, HLRZ, World Scientific, p. 137.

Lawrence, S., Giles, C. L. and Tsoi, A. C. (1996). What size neural network gives optimal generalization? Convergence properties of backpropagation, *Technical Report CS-TR-3617*, Department of Electrical and Computer Engineering, University of Queensland, St. Lucia 4072, Australia.

Le Cun, Y., Denker, J. and Solla, S. (1989). Optimal brain damage, *Advances in Neural Information Processing Systems* **2**: 598–605.

Leen, T. K. (1995). From data distribution to regularization in invariant learning, *Neural Computation* **7**: 974–981.

Lenz, R. (1990). Group invariant pattern recognition, *Pattern Recognition* **23**(1/2): 199–217.

Li, C. and Wu, C.-H. J. (1993). Introducing rotation invariance into the Neocognitron model for target recognition, *Pattern Recognition Letters* **14**: 985–995.

Li, S. (1992). Matching: Invariant to translations, rotations and scale changes, *Pattern Recognition* **25**(6): 583–594.

Li, X. and Roeder, N. (1994). Experiments in detecting face contours, *Vision Interface Conference*.

Lin, F. and Brandt, R. D. (1993). Towards absolute invariants of images under translation, rotation and dilation, *Pattern Recognition Letters* **14**: 369–379.

Mathews, J. and Walker, R. (1970). *Mathematical Methods of Physics*, 2 edn, Addison-Wesley Publishing Company, Inc.

McGraw, G. (1992). Letter Spirit: Recognition and creation of letterforms based on fluid concepts, *Technical report*, Department of Computer Science, Indiana University, Bloomington, Indiana 47405.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A. and Teller, E. (1953). Equation of state calculations by fast computing machines, *Journal of Chemical Physics* **21**(6): 1087–1092.

Minsky, M. and Papert, S. (1969). *Perceptrons. An introduction to computational geometry*, The MIT Press, Cambridge, London.

Miyake, S. and Fukushima, K. (1984). A neural network model for the mechanism of feature-extraction, *Biological Cybernetics* **50**: 377–384.

Moody, J. E. (1992). The *effective* number of parameters: An analysis of generalization and regularization in nonlinear learning systems, *Advances in Neural Information Processing Systems* **4**: 847–854.

Mundy, J. L., Zisserman, A. and Forsyth, D. (eds) (1993). *Applications of Invariance in Computer Vision*, Lecture Notes in Computer Science, Springer-Verlag, Berlin.

Nowlan, S. and Hinton, G. (1992a). Simplifying neural networks by Soft-Weight Sharing, *Neural Computation* **4**(4): 473–493.

Nowlan, S. J. and Hinton, G. E. (1992b). Adaptive Soft Weight Tying using gaussian mixtures, *Advances in Neural Information Processing Systems* **4**: 993–1000.

Ohlsson, M. (1992). Extensions and explorations of the elastic arms algorithm, *Technical Report LU TP 92-28*, Department of Theoretical Physics, University of Lund, Sölvegatan 14A, S-22362 Lund, Sweden.

Omlin, C. W. and Giles, C. L. (1992). Training second-order recurrent neural networks using hints, *in* D. Sleeman and P. Edwards (eds), *Machine Learning: Proceedings of the Ninth International Conference*, Morgan Kaufmann, San Mateo, CA.

Perantonis, S. J. and Lisboa, P. J. (1992). Translation, rotation, and scale invariant pattern recognition by higher-order neural networks and moment classifiers, *IEEE Transactions on Neural Networks* **3**(2): 241–251.

Pintsov, D. A. (1989). Invariant pattern recognition, symmetry, and radon transforms, *Journal of the Optical Society of America (A)* **6**(10): 1544–1554.

Pitts, W. and McCulloch, W. S. (1947). How we know universals: the perception of auditory and visual forms, *Bulletin of Mathematical Biophysics* **9**: 127–147.

Plaut, D. and Hinton, G. (1987). Learning sets of filters using backpropagation, *Computer Speech and Language* **2**: 35–61.

Prem, E., Mackinger, M. and Dorffner, G. (1993). Concept support as a method for programming neural networks with symbolic knowledge, *Proceedings of the German Artificial Intelligence Conference*, Springer Verlag, Berlin-Heidelberg.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P. (1992). *Numerical Recipes in C : The Art of Scientific Computing*, 2 edn, Press Syndicate of the University of Cambridge, Cambridge, U.K.

Redding, N. J., Kowalczyk, A. and Downs, T. (1993). Constructive higher-order network algorithm that is polynomial time, *Neural Networks* **6**: 997–1010.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain, *Psychological Review* **65**: 386–408.

Rosenfeld, A. and Kak, A. C. (1982). *Digital Picture Processing*, Academic Press, Orlando, FL.

Rubinstein, J., Segman, J. and Zeevi, Y. (1991). Recognition of distorted patterns by invariance kernels, *Pattern Recognition* **24**(10): 959–967.

Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1986a). Learning representations by back-propagating errors, *Nature* **323**: 533–536.

Rumelhart, D., Hinton, G. and Williams, R. (1986b). Learning internal representations by error propagation, *in* D. Rumelhart and J. McClelland (eds), *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, Vol. 1, MIT Press, Cambridge, MA, pp. 318–362.

Salas, S., Hille, E. and Anderson, J. T. (1986). *Calculus: One and several variables, with analytic geometry*, fifth edn, John Wiley & Sons, New York.

Sarle, W. S. (1994). Neural networks and statistical models, *Proceedings of the Nineteenth Annual SAS Users Group International Conference*.

Schmidt, W. A. and Davis, J. P. (1993). Pattern recognition properties of various feature spaces for higher order neural networks, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **15**(8): 795–801.

Segman, J., Rubinstein, J. and Zeevi, Y. Y. (1992). The canonical coordinates method for pattern deformation: Theoretical and computational considerations, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **14**(12): 1171–1183.

Simard, P., Victorri, B., Cun, Y. L. and Denker, J. (1992). Tangent Prop - a formalism for specifying selected invariances in an adaptive network, *Advances in Neural Information Processing Systems* **4**: 895–903.

Sontag, E. D. (1992). Feedback stabilization using two-hidden-layer nets, *IEEE Transactions on Neural Networks* **3**: 981–990.

Spirkovska, L. and Reid, M. B. (1994). Higher-order neural networks applied to 2D and 3D object recognition, *Machine Learning* **15**(2): 169–199.

Squire, D. M. and Caelli, T. M. (1995). Shift, rotation and scale invariant signatures for two-dimensional contours, in a neural network architecture, *to appear in the Proceedings of the 1st International Conference on the Mathematics of Neural Networks and Applications (MANNA 95), Lady Margaret Hall, Oxford, published as a special edition of Annals of Mathematics and Artificial Intelligence*, J.C. Baltzer Scientific Publishing Company, Basel - Switzerland.

Srinivasa, N. and Jouaneh, M. (1992). A neural network model for invariant pattern recognition, *IEEE Transactions on Signal Processing* **40**(6): 1595–1598.

Srinivasa, N. and Jouaneh, M. (1993). An invariant pattern recognition machine using a modified ART architecture, *IEEE Transactions on Systems, Man and Cybernetics* **23**(5): 1432–1437.

Srinivasan, V., Bhatia, P. and Ong, S. (1994). Edge detection using a neural network, *Pattern Recognition* **27**(12): 1653–1662.

Tebelski, J. and Waibel, A. (1990). Large vocabulary recognition using linked predictive networks, *IEEE Proceedings of the 1990 International Conference on Acoustics, Speech and Signal Processing*, IEEE.

Vanderkooy, G. E. (1996). Line matching for uncalibrated cameras using projective invariants, *Technical report*, Vision and Electronic Measurement Laboratory, Department of Mechanical Engineering, University of Victoria, Canada.

Waibel, A., Jain, A., McNair, A., Saito, H., Hauptmann, A. and Tebelski, J. (1991). JANUS: A speech-to-speech translation system using connectionist and symbolic processing strategies, *IEEE Proceedings of the 1991 International Conference on Acoustics, Speech and Signal Processing*, IEEE.

Wechsler, H. (1990). *Computational Vision*, Academic Press Inc., 1250 Sixth Avenue, San Diego, CA 92101.

Weiss, I. (1993a). Geometric invariants and object recognition, *International Journal of Computer Vision* **10**(3): 207–231.

Weiss, I. (1993b). Noise-resistant invariants of curves, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **15**(9): 943–948.

Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits, *IRE WESCON Convention Record*, IRE, New York, pp. 96–104.

Wiles, J. and Elman, J. (1995). Learning to count without a counter: A case study of dynamics and activation landscapes in recurrent networks, *Proceedings of the Seventeenth Annual Conference of the Cognitive Society*, MIT Press, Cambridge, MA.

Wiles, J. and Ollila, M. (1993). Intersecting regions: The key to combinatorial structure in hidden unit space, *Advances in Neural Information Processing Systems* **5**: 27–33.

Winther, O., Lautrup, B. and Zhang, J.-B. (1995). Optimal learning in multilayer neural networks, *Technical Report 95-200*, CERN, Theory Division, 1211 Genève 23, Switzerland.

Zetzsche, C. and Caelli, T. (1989). Invariant pattern recognition using multiple filter image represtations, *Computer Vision, Graphics and Image Processing* **45**: 251–262.