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.

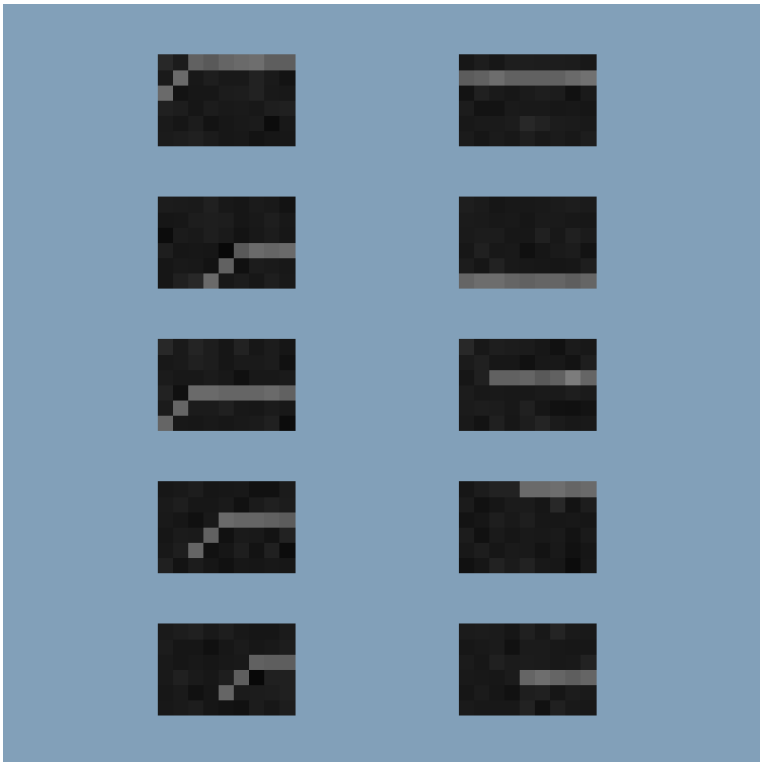


Figure 4.1: Examples of Risers (left) and Non-risers (right) from the Plaut and Hinton (1987) classification problem. Signals were to be classified as R or NR as a function of their spectrogram shapes. These are the training patterns used in the simulations in this thesis.

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

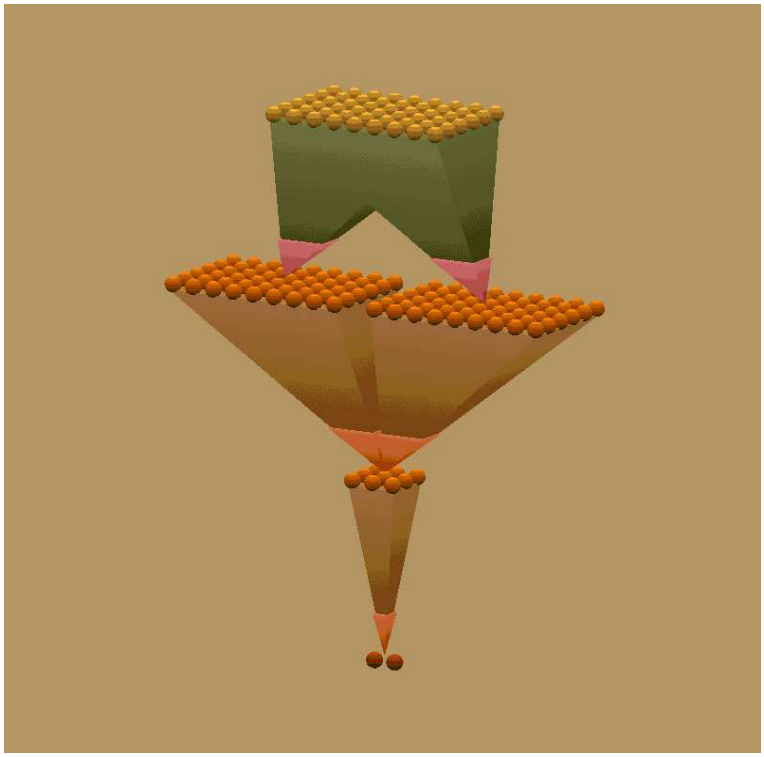


Figure 4.2: The MBNN used to solve the riser/non-riser classification problem in this study. Input (top) to second level connections are modeled by Gabor (M.3)-type connections. Other connections are defined by M.1 connections.

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

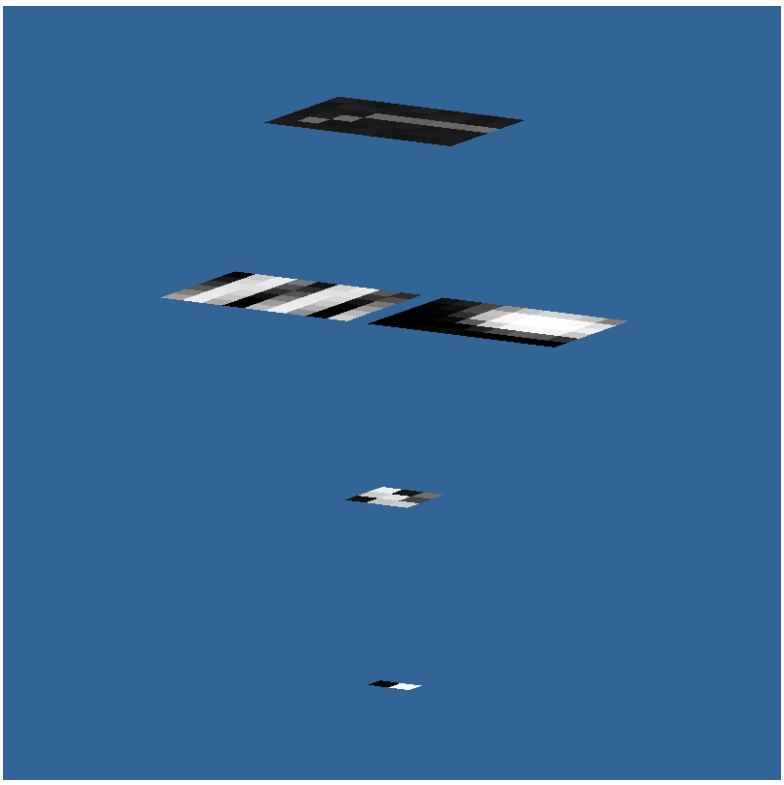


Figure 4.3: Response of MBNN to a riser. Note that the left-hand Gabor layer is responding to the vertical component of the pattern.

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

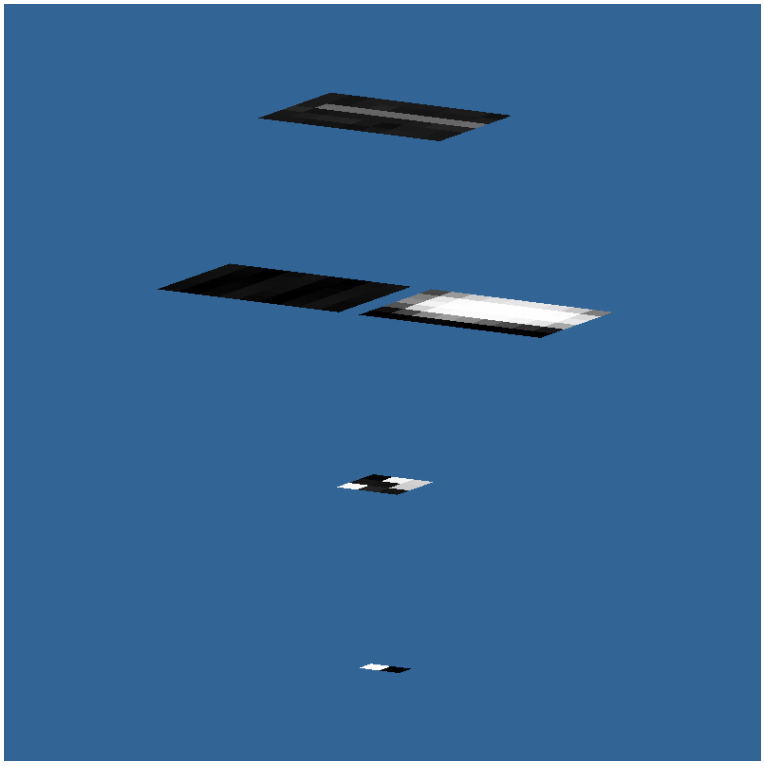


Figure 4.4: Response of MBNN to a non-riser.

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.

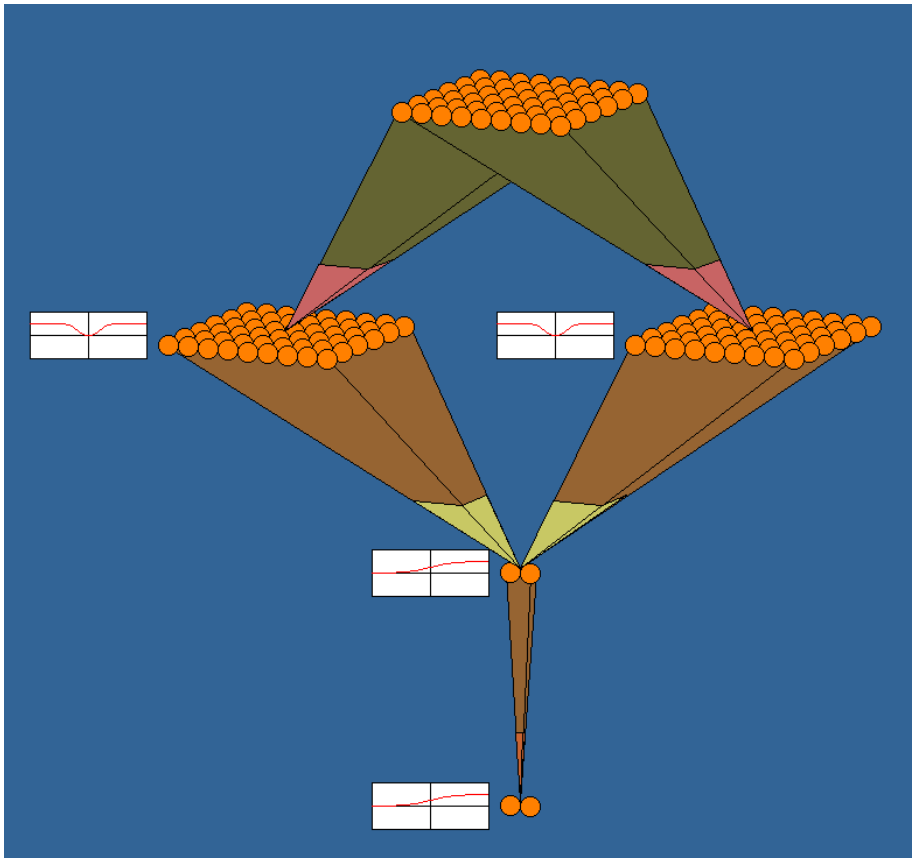


Figure 4.5: A MBNN that solves the Riser/Non-riser problem with only 22 parameters.

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.

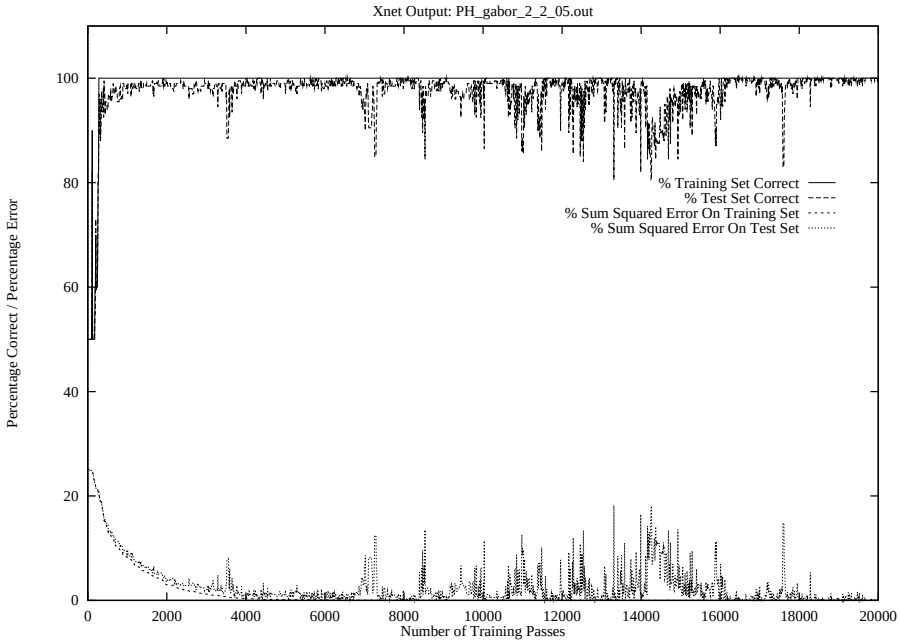
The use of simulated annealing for training means that the non-monotonic derivatives of the Gabor functions do not prevent the filter orientations from changing significantly during training. It should also be noted that using simulated annealing results in very much faster training, since no derivatives 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.

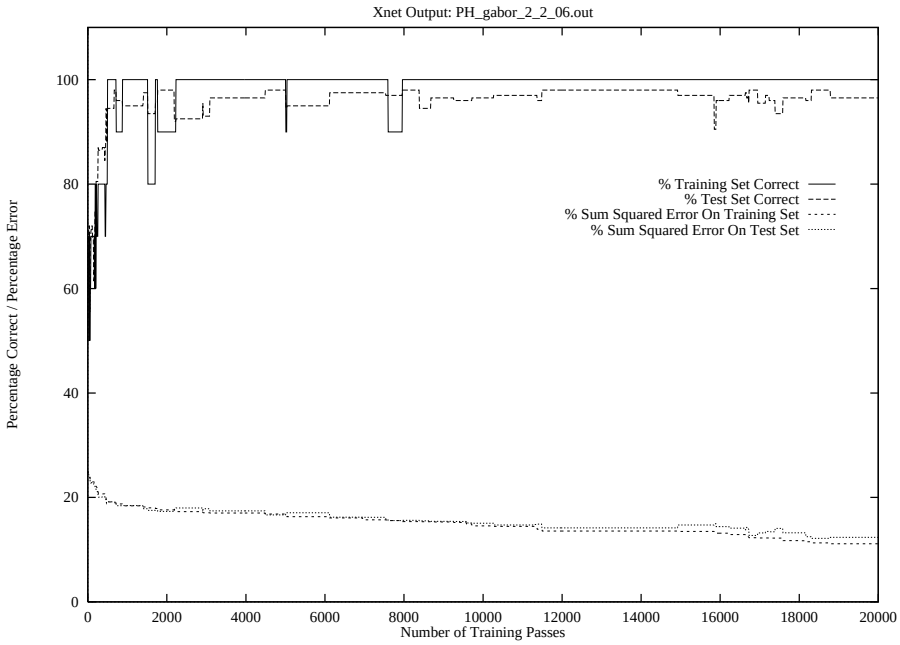
	Training Data	Test Data
Network 1	11.05476	16.26678
Network 2	0.00015	0.49714
Network 3	10.39662	15.41420
Network 4	0.00006	1.27624
Network 5	0.00079	0.06715
Network 6	11.13610	12.36650
Network 7	0.00024	0.30148
Network 8	0.00016	0.00016
Network 9	0.00058	4.00381
Network 10	0.89500	1.37818

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

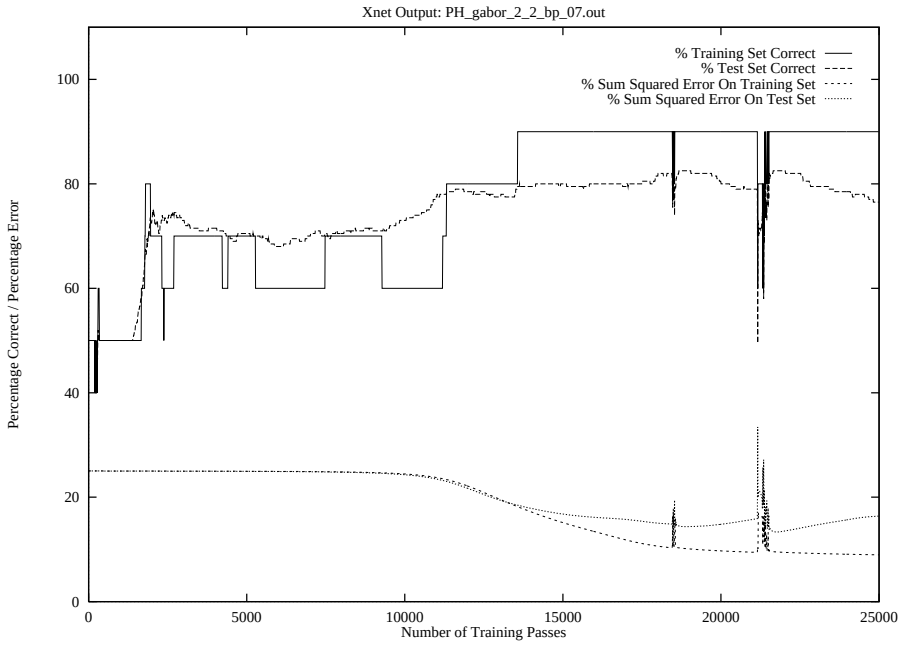


Figure 4.7: Catastrophic failure during training of a 22 parameter MBNN with backpropagation.

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

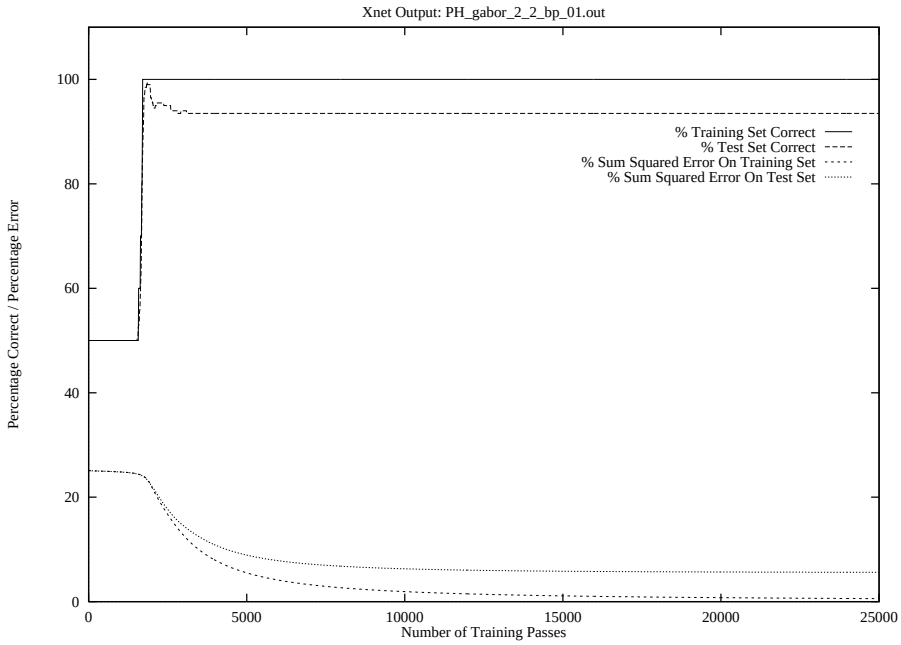


Figure 4.8: Overfitting of training data, resulting in diminished test set performance during training of a 22 parameter MBNN with backpropagation.

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