Model-Based Neural Networks

Terry M. Caelli* David McG. Squire† Tom P. J. Wild

The University of Melbourne
Melbourne, AUSTRALIA
Neural Networks, *Vol. 6, pp. 613-625, 1993*
Received 21 October 1991, Revised and Accepted 26 October 1992

26 October 1992

In this paper we show how neural networks can be formulated in terms of various parameterised connection models which explicitly encode desired properties of the target system. Such a modeling approach to neural networks raises issues about their relationships to other technologies such as Adaptive Filtering and Principal Components Analysis. The benefits of this approach can be a significant decrease in the parameter space, improved generalisation, and a learning procedure which guarantees a priori specified invariance constraints.

1 Introduction

Many claims have been made about 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 in the brain. Added to this, there is also 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 paper is not aimed a challenging these claims, but a considering fundamental computational problems of NNs related to their usual representational dimensionality and model adequacy. In particular, this paper is concerned with the problem of integrating domain/task specific knowledge into the very architecture of NNs.

It is important to note here, then, that most NN formulations have a few central

*This project was funded by a grant from the Key Centre for Knowledge Based Systems and the CRC: Centre for Intelligent Decision Systems.

†David Squire was funded by an Australian Postgraduate Research Award

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 to model it since the NN learns the appropriate weights which are necessary and sufficient to predict behaviour.

4. The degree of nonlinearity is usually the same between all layers and is chosen arbitrarily.

That is, most applications of NNs are based upon the assumption that "solutions" to problems come about by 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" into 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.

Our aim, in this paper, is to formalise an emerging perspective to NNs (see, for example, [1]; [2]) which, while retaining the essentials of the NN approach, enables further modeling of the system. This results in dramatically improved generalisation of classification performance to patterns not present in the training data, and representation in considerably lower dimensional state spaces.

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

1.1 The Classical Summation Neural Network

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

$$x_j = \sum_i y_i w_{ij} \tag{1}$$

and the output is defined by the logistic function

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

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 feedforward architecture, as shown in Figure 1 (as, for example, employed by [3]). 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 and given one (see Figure 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 input output (I-O) characteristics (equations 1 and 2 above).

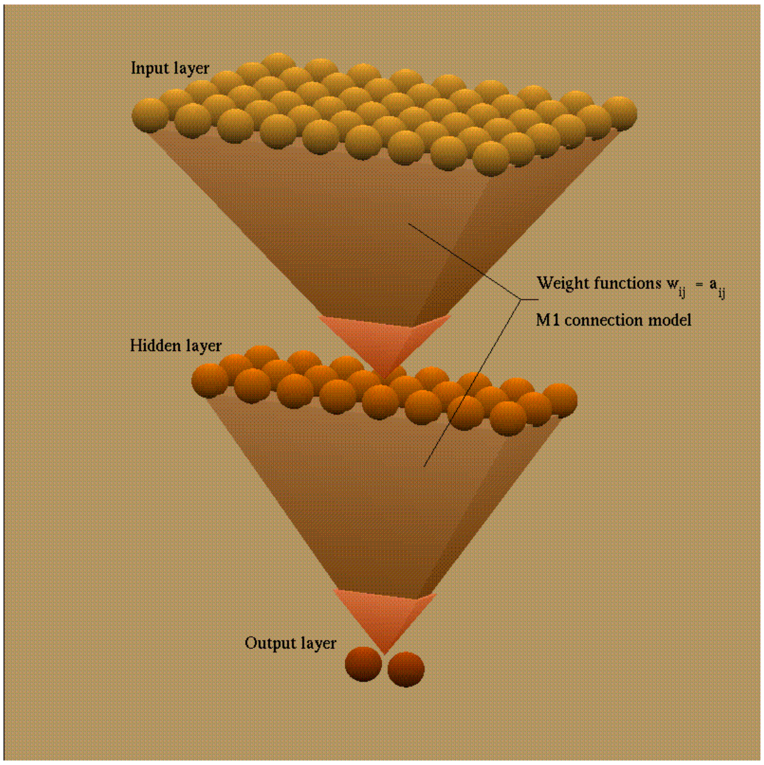

Figure 1: A classical summation network. The weighting function between all pairs of levels is of type M.1.

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 w_{ij}s such that a given output error, or "cost", function E is minimised:

$$\min_{w_{ij}} \left\{ E = \frac{1}{2} \sum_j \sum_c (y_{jc} - t_{jc})^2 \right\} \tag{3}$$

where w_{ij} refers to all connection weights over all levels of the network, c to the number of classes, y_{jc} to actual output, and t_{jc} to desired output.

Though described in a variety of ways, learning techniques use either deterministic gradient descent (in particular, "backpropagation") or stochastic relaxation methods (see, for example, [4]) to find a set of weights (a state of the network) that satisfies Equation 3. Solution or convergence times can be improved with various acceleration techniques. However, this paper is 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 3 for the training examples, but do not characterise 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 minimisation or search for global minima of Equation 3. Here, we propose to retain the notion that states can certainly 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. However, to put this view in perspective we will first relate the parameter estimation problem in NNs to similar problems in Signal Processing and Principal Components Analysis.

1.2 A Matrix View of Neural Networks

We first note that, for a given set of connections, Equations 1 and 2 can be written in matrix form as:

$$X_n = g\left\{\mathbf{W} Y_m\right\} \tag{4}$$

where g corresponds to the logistic nonlinear transducer (Equation 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, $n < m$. Here, the nonlinear transducer is used to amplify strong inputs and decrease the responses to weak inputs. The net effect is that the NN procedure endeavours to discover a mapping which satisfies an overt constraint like least squares (Equation 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" (dimensions in the n-dimensional range map) correspond to components of the eigenstructure in the mappings, albeit constrained by the optimisation goal. Except for the nonlinear transducer function, then, the use of cascaded 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 [5], [6].

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 maximise the between-class and minimise the within-class variance from their projections onto such planes. Discriminant functions are linear equations of the form:

$$z = \sum_{i=1}^{m} a_i y_i + a_0 \tag{5}$$

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 5 generalises to the matrix form:

$$Z = \mathbf{D}Y \tag{6}$$

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.

Similar formulations for classification have occurred in the Pattern Recognition literature where, for example, the Least Squares Minimum Distance Classifier [6] 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 satisfying 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 characterise the task or to have desired invariances, rather than hoping that the training data will cause the optimisation technique to find a set of weights with these properties. Secondly, this allows the dimensionality of the system to be reduced, which reduces the chance of finding a local minimum that characterises 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 the roles of some hidden units, or of whole hidden layers, in advance. Thus the Supervisor's knowledge of which components 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.

2 The Model-Based Classifier

In this implementation of MBNNs, the networks are modeled in two senses. First, the multilayer feedforward structure of TNNs (Traditional Neural Networks, without modeling the connections) is generalised to a multilevel feedforward structure, where there are one or more layers of nodes at each level. An example of such a structure appears in Figure 3. This allows, for instance, 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, and more powerful, sense in which the MBNN is constructed to model the task which it is meant to perform is that the connection weights between nodes can be made to 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 parametrically 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 layer, such that the distance between horizontally or vertically adjacent nodes in 1. The origin of coordinates for each layer is situated at the centroid of the layer. the distance between nodes i and j, d_{ij}, is considered to be the usual Euclidean distance between the projection of node i onto the layer in which node j is situated, and node j. In practice, this means that the distance between nodes is calculated using their coordinates as if they were in the same layer. In this sense, the actual positions of nodes play an important role in this form of MBNN.

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:

$$w_{ij} = f\left(\mathbf{a}_{ij}, d_{ij}\right). \tag{7}$$

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

$$w_{ij} = f\left(\mathbf{a}_{j}, d_{ij}\right). \tag{8}$$

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

$$w_{ij} = f\left(\mathbf{a}, d_{ij}\right). \tag{9}$$

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 17), 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 order of differentiation and 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 more robustness to noise and will generalise 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 term in the Supervised Learning Process (Equation 3). That is,

$$\frac{\partial E}{\partial y_i} = \sum_c (y_{jc} - t_{jc}) \tag{10}$$

and

$$\frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial x_j} \tag{11}$$

For our first implementation of MBNN we have modeled connection weights w_{ij} as a function of a vector of parameters, $\mathbf{a}_{ij}$, and the distance d_{ij} between the nodes:

$$w_{ij} = f(\mathbf{a}_{ij}, d_{ij}). \tag{12}$$

In fact, the usual NN model is a degenerate case of Equation 12 above where

$$w_{ij} = a_{ij} \tag{13}$$

Three popular weighting functions are: one, a gaussian weighting function:

$$w_{ij} = a_1 \exp\left(-a_2 d_{ij}^2\right), \tag{14}$$

two, the Gabor function:

$$w_{ij} = a_1 \exp\left(-a_2 d_{ij}^2\right) \cos\left(a_3 x + a_4 y + a_5\right), \tag{15}$$

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

$$w_{ij} = \cos\left(a_1 d_{ij}\right), \tag{16}$$

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

$$\frac{\partial E}{\partial a_k} = \frac{\partial E}{\partial w_{ij}} \frac{\partial w_{ij}}{\partial a_k} \tag{17}$$

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

$$\frac{\partial E}{\partial a_1} = \exp\left(-a_2 d_{ij}^2\right) \tag{18}$$

and

$$\frac{\partial E}{\partial a_2} = -a_2 d_{ij}^2 \exp\left(-a_2 d_{ij}^2\right). \tag{19}$$

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

$$a_k(t+1) = a_k(t) - \varepsilon \frac{\partial E}{\partial a_k(t)} - \alpha \frac{\partial E}{\partial a_k(t-1)} \tag{20}$$

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

2.1 Relation to Adaptive Filtering

For this specific formulation, the MBNN essentially becomes a generalisation of the well-known filter estimation problem in the 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 convolution function, defining the action of a filter $h(x, y)$ on a signal $I(x, y)$ (here, 2-dimensional), is defined by:

$$R(x_1, y_1) = \sum_{uv} h(u, v) \cdot I(x_0 + u, y_0 + v) \tag{21}$$

where (x_0, y_0) and (x_1, y_1) correspond to input and output (image) planes. 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). The output coordinate system in Equation 21 corresponds to the destination layer in the traditional NN. Case M.1 corresponds to the highest form of nonstationarity where the parameters $\mathbf{a}$ of the kernel are a function of the input and output position pairs:

$$w(u, v) \equiv w(u, v : \mathbf{a}(x_0 + u, y_0 + v; x_1, y_1)). \tag{22}$$

Case M.2 corresponds to:

$$w(u, v) \equiv w(u, v : \mathbf{a}(x_1, y_1)). \tag{23}$$

and, finally, case M.3 corresponds to:

$$w(u, v) \equiv w(u, v : \mathbf{a}). \tag{24}$$

Case M.2 corresponds to a "generalised" Wigner function [7] 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 (see [7] for examples). However, the aim of this investigation was to study how such structures , in their most general forms, can be formulated in the *same architecture* as the remainder of the NN.

3 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.

A typical TNN approach to the invariance problem is to include shifted, scaled or rotated versions of patterns in the training set (see, for example, [8]). 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 the 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 [9] differs from this approach in that it augments the usual least squares cost function with a 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 distortions of the input image corresponding to the action of that operator. Tangent Prop is thus close to MBNNs in its aim: to include *a priori* knowledge of the desired invariances of the network, rather than requiring a training set that is sufficiently varied to cause such invariances to arise spontaneously. It does not, however, place any 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 "stripes" 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 stripes of only one orientation. Consequently, the network's response must be invariant under rotations 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 16) to two 15×15 hidden layers, which were, in turn, connected using M.2 simple weights to a 2×2 hidden layer, which was 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 to 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 to small 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 cosine grating. The activation $I(x, y)$ at node (x, y) of the input layer was given by:

$$I(x,y) = \frac{1}{2}K\left(1 + \cos\left(2\pi\left(u\frac{x}{x_{\max}} + v\frac{y}{y_{\max}}\right)\right)\right). \tag{25}$$

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 normalising 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 quantisation errors. K is found by integrating over the input layer with respect to x and y:

$$K = \left(1 - \frac{\cos\left(2\pi\left(u+v\right)\right) + 1 - \cos\left(2\pi u\right) - \cos\left(2\pi v\right)}{4\pi^2 uv}\right). \tag{26}$$

Note that this correction is necessary only when u or v in not an integer. The reason for using normalised 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:

$$f = \sqrt{u^2 + v^2}, \tag{27}$$

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

$$W = \frac{5}{\text{number of inputs to node}}. \tag{28}$$

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 optimisation techniques have trouble with such "saturated" nodes also, since small changes in the input weights cause no appreciable change in the output.

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 initialised 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 is sinusoidal. Gradient descent is consequently an inappropriate optimisation technique for this problem. Whilst this is to some extent a disadvantage, it must be remembered that local minima are a (often left unmentioned) problem for TNNs also. The fact that the error surfaces for the MBNNs implementing particular invariances are not monotonic may indicate that such systems of weights are not learnable using backpropagation on TNNs.

The optimisation technique used was simulated annealing [10]. The following cost function was used for the network's performance on the training patterns:

$$c = \frac{\lambda \frac{1}{2}\left(\frac{\sinh(\kappa(1-\Psi))}{\sinh(\kappa)}+1\right)+\frac{\sum_j \sum_c (y_{jc}-t_{jc})^2}{jc}}{1+\lambda}. \tag{29}$$

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

$$\frac{1}{2}\left(\frac{\sinh\left(\kappa\left(1-\Psi\right)\right)}{\sinh\left(\kappa\right)}+1\right) \tag{30}$$

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 penalised 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. The second term is just the squared error (as in Equation 3) normalised 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 normalised 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 29) 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 Metropolis simulated annealing algorithm [10]. 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 Metropolis 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. 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 1.

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
Test Avg. $\pm\sigma$	53 ± 12	Test Avg. $\pm\sigma$	97 ± 7.0

Table 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, generalise to the rotated test patterns extremely well. In fact, in most cases, they generalise perfectly. The TNNs, on the other hand, to within one standard deviation, do not generalise 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 unfeasible.

4 The Plaut and Hinton Study

To further illustrate the differences between this model and more traditional ones, we have considered the problem of classifying rising and nonrising signals in speech spectrograms, as studied by Plaut and Hinton [3]. 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 consisted of classifying as risers or nonrisers synthetic spectrograms, represented as patterns on a 6×9 grid, where the vertical direction corresponded to frequency, and the horizontal direction to sampling time. The patterns were generated under the following constraints:

1. Generate Risers (R) and Nonrisers (NR) with equal probability.
2. Randomly choose one of the six possible frequencies for Nonrisers. 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 2.

This procedure generates a population (neglecting noise cases) with 50 members: 20 risers and 30 nonrisers. Plaut and Hinton 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.

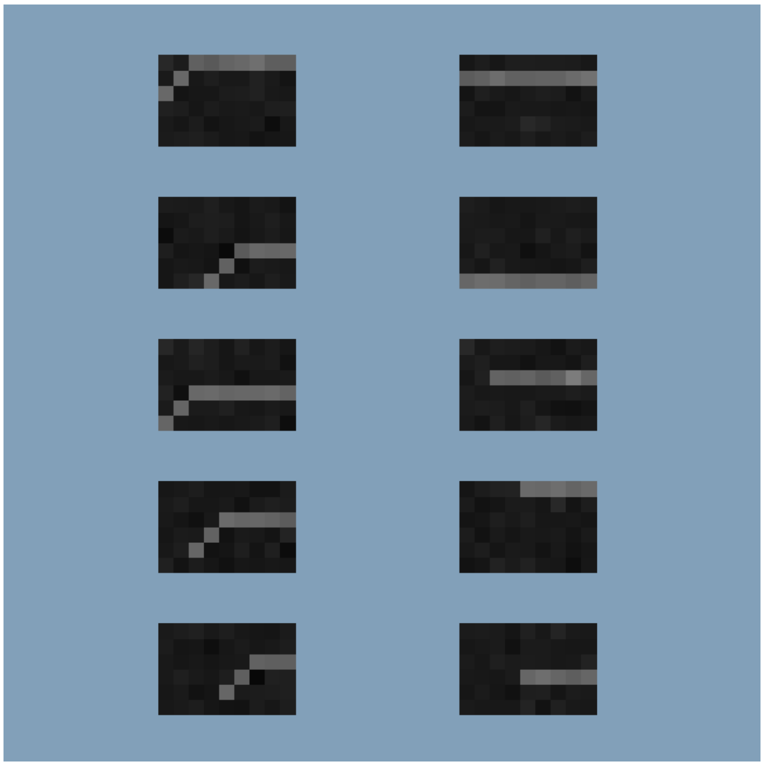

Figure 2: Examples of risers (R - left) and nonrisers (NR - right) from the Plaut and Hinton (1987) classification problem. Signals were to be classified as R or NR as a function of their spectrogram shapes. These are examples of training patterns used in this study.

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.

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 9 and Gabor weight functions (see Equation 15), 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 3.

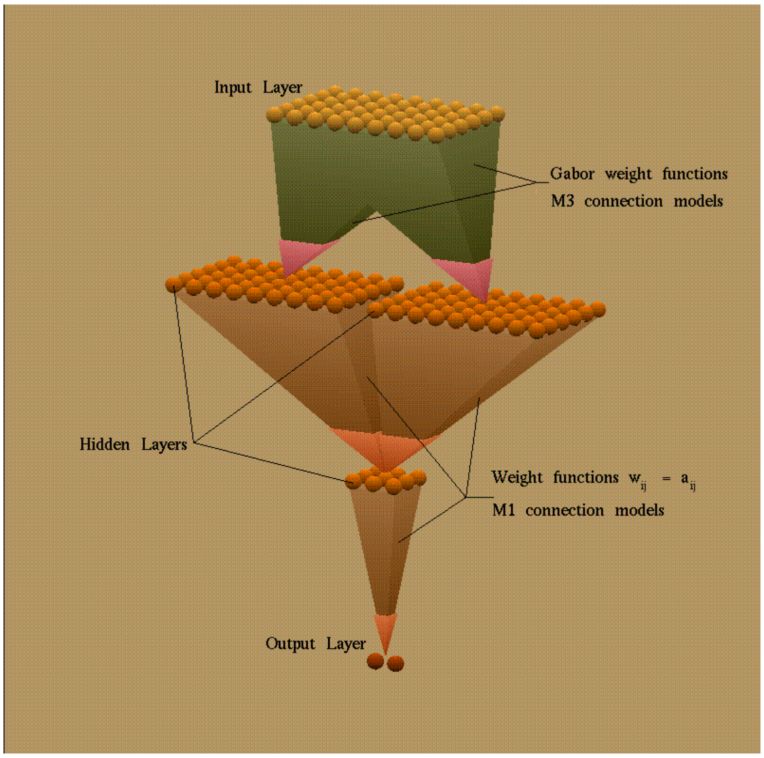

Figure 3: The MBNN used to solve the riser/nonriser classification problem in this study. Input to the first hidden level connections are modeled by Gabor (M.3)-type connections. Other connections are defined by M.1 connections.

A rectifying transfer function of the form:

$$y = \frac{1 - e^{-x^2}}{1 + e^{-x^2}} \tag{31}$$

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 (normalisation) form, as shown in Equation 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 even 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 anyway. Such a "self-organisation" 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 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 generalise 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 nonrisers, which were initially selected at random from the population described above (see Figure 2). The parameters were adjusted after each pass through this set. Also, *after each pass*, the performance of the network was measured, both in terms of the percentage of the training set correctly classified, and the percentage classified of a test set, consisting of 100 risers and 100 nonrisers not used in the training process.

Both kinds of network achieved 100% correct classification of 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 generalise.

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 generalise.

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 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 2.

Figures 4 and 5 show the responses of one of the MBNNs to a riser and a nonriser, respectively, and the output layers show that the two patterns have been classified cor-

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

Table 2: Classification Performance (Percent Correct) of Networks on 200 Novel Patterns after 2000 Training Iterations on 10 Patterns

rectly. 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 initialised 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 15) 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 filter has "learnt" to respond to that feature of risers that is *orthogonal* to nonrisers. This illustrates the analogy between the network's function and principal components analysis (see Section 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.

5 Comparison With Other Techniques For Improving Generalisation

Previous efforts to improve the generalisation performance of NNs on novel data differ from the MBNN approach in a fundamental way. Only one of the other techniques reviewed 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 characterise 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 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

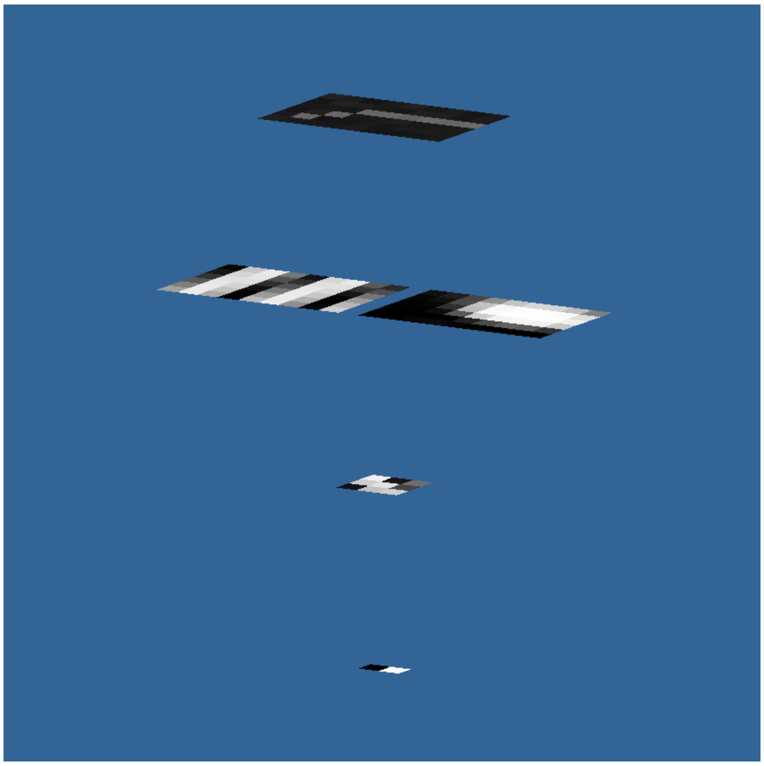

Figure 4: Response of MBNN to a riser. Note that the left-hand Gabor layer is responding to the vertical component of the pattern.

is to remove the need t have a training set that spans the population of possible input patterns, requiring only that the training set span the population in the feature space determined by the explicit modeling of the network.

The most common method used to attempt to improve generalisation is to augment the usual least squares cost function E (Equation 3) with some auxiliary function of the weights:

$$C = \frac{1}{2} \sum_{j} \sum_{c} \left(y_{j\,c} - t_{j\,c} \right)^2 + \lambda g \left(w_{ij} \right). \tag{32}$$

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 minimising Equation 3 and minimising 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 generalise better. 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.

Probably the most common auxiliary function is the sum of the squares of all the

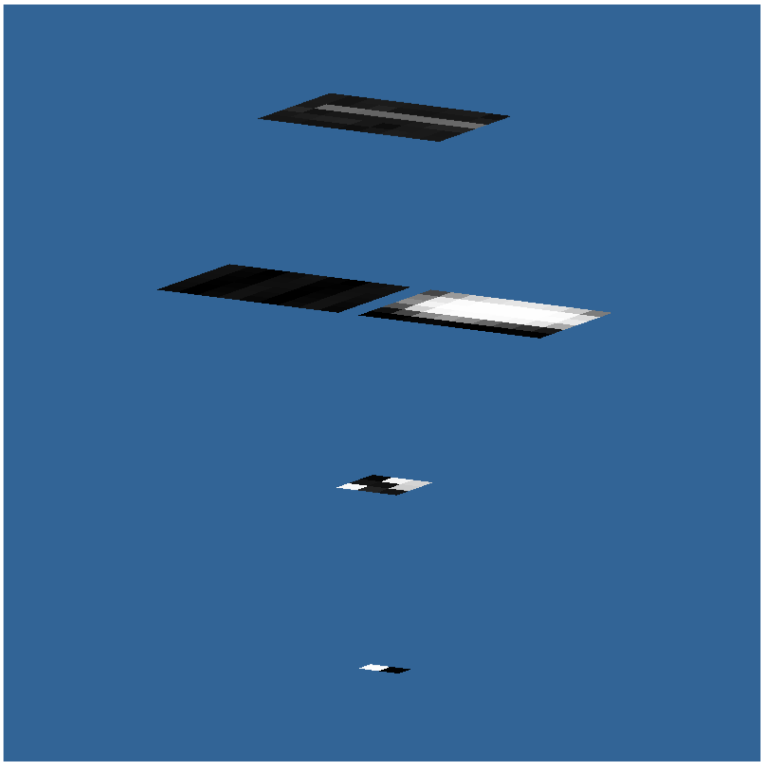

Figure 5: Response of MBNN to a nonriser.

weights in the network [11] [3]:

$$C = \frac{1}{2} \sum_j \sum_c (y_{j\,c} - t_{j\,c})^2 + \lambda \frac{1}{2} \sum_i w_{ij}^2. \tag{33}$$

This cost function leads to a simple "weight decay" term in the weight update equation (Equation 20):

$$w_k(t+1) = w_k(t) - \varepsilon \frac{\partial E}{\partial w_k(t)} - \lambda w_k. \tag{34}$$

Where E is the sum of squared errors from Equation 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 minimisation 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 minimises the sensitivity of the output to noise in the input [12], 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".

Soft Weight-Sharing, proposed by Nowlan and Hinton [12], is, in essence, a generalisation of this 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. An attempt was made to implement Soft Weight-Sharing so that its generalisation performance could be compared to that of MBNNs on the riser/nonrisers 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).

Another technique that is claimed to improve generalisation is Cascade-Correlation [13]. 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 [9] 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. If the response of the network is invariant under some distortion (e.g., rotation, translation, expansion) of the input pattern, then the derivative of its output with respect to such distortions should be zero. Tangent Prop augments the usual least squares error function with a term that is proportional to the square of the error of the derivatives of the outputs with respect to the known desired invariances. This 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 generalisation 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. 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 cosine grating problem appear in Table 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 generalisation performance for this task. The generalisation performance of all these backpropagation trained TNNs, with and without weight decay, is still well below that of the MBNNs described in Section 3.

	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
Average $\pm\sigma$	63 ± 4.5	62 ± 5.7	60 ± 7.1	54 ± 6.5

Table 3: Classification Performance (Percent Correct) of TNNs on Cosine Grating 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/nonriser problem appear in Tables 4 and 5. Again it was not possible to claim that weight decay improved generalisation. When $\lambda = 0.005$, the performance on the training 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. Again the performance of the MBNNs described in Section 4 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
Average $\pm\sigma$	100.0 ± 0.00	90.0 ± 0.00	50.0 ± 0.00

Table 4: Classification Performance (Percent Correct) of TNNs on Riser/Nonriser Training 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 generalisation performance.

It can be seen, therefore, that MBNNs offer a superior means of achieving good generalisation 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 generalisation 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.

	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
Average $\pm\sigma$	50.0 ± 0.00	53.5 ± 0.00	50.0 ± 0.00

Table 5: Classification Performance (Percent Correct) of TNNs on Riser/Nonriser Test Data for Various Values of the Weight Decay Parameter λ, After 1000 Iterations

6 Discussion

The main feature of MBNNs, over traditional networks, lies in the modeling or parameterising the connections to satisfy specific constraints, which model the task that the NN is designed to perform. We have illustrated this in a classification problem where it seemed appropriate to model the connections between the initial layers in terms of orientation-specific weighting functions whose actual response properties (preferred orientation responses) were learned during the training phase. Though such an approach is closely related to the theory of Adaptive Filters, the benefit of NNs is that they permit a generalisation of the usual shift-invariance, or superposition, property such that we can model the inhomogeneities of processing over different regions of the input signal. The benefits of the MBNN system are that it permits solutions in potentially much lower dimensional state spaces, allows us to model the types of invariances required in the processing or recognition of the signal, and is, apparently, better at generalisation.

This is not the first attempt to combine NNs with Adaptive Filter Theory, or even the modeling of connection weights. For example Waibel et. al. [2] have already examined the shift-invariant property for speech recognition and Wechsler [7] has considered the case of scale and rotation invariant pattern recognition. What we have tried to do here, however, is to organise the various types of model-based systems in terms of transmitter-receiver indexing and to parameterise the network while *preserving the same architecture.* The system is particularly useful for coping with invariant pattern recognition in so far as the connection function can be chosen to reflect the required invariance. For example, rotation, shift and scale invariance encoding could involve the use of hidden layers whose inputs came from connection functions which are log-polar (scale and rotation; [14]) and M.3 type (rotation, scale and shift invariant; [15]).

Though not applied here, the additional training of the transducer function (parameterised exponent form of Equation 2) serves a significant role in MBNNs as this extra parameter, for each pair of layers, determines the degree of nonlinearity involved in solving a given learning problem. Informal results with this have consistently demonstrated that the degree of nonlinearity, at least for classification problems, increases as we progress through the network. This, and the complete Model-Based approach, argues for applications of NNs bringing knowledge of the system and known constraints on the solution space to the estimation of connection weights.

References

[1] J. Tebelski and A. Waibel. Large vocabulary recognition using linked predictive networks. In *IEEE Proceedings of the 1990 International Conference on Acoustics, Speech and Signal Processing*. IEEE, 1990.

[2] A. Waibel, A. Jain, A. McNair, H. Saito, A. Hauptmann, and J. Tebelski. Janus: A speech-to-speech translation system using connectionist and symbolic processing strategies. In *IEEE Proceedings of the 1991 International Conference on Acoustics, Speech and Signal Processing*. IEEE, 1991.

[3] D.C. Plaut and G.E. Hinton. Learning sets of filters using backpropagation. *Computer Speech and Language*, 2:35–61, 1987.

[4] E. Aarts and J. Korst. *Simulated annealing and Boltzmann machines*. John Wiley and Sons, New York, 1989.

[5] F. Ade. Characterisation of textures by "eigenfilters". *Signal Processing*, 5:451–457, 1983.

[6] N. Ahmed and K. Rao. *Orthogonal transforms for digital signal processing*. Springer-Verlag, New York, 1975.

[7] H. Wechsler. *Computational Vision*. Academic Press, New York, 1990.

[8] K. Fukushima, S. Miyake, and T. Ito. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5):826–834, 1983.

[9] P. Simard, B. Victorro, Y. Le Cun, and J. Denker. Tangent prop - a formalism for specifying selected invariances in an adaptive network. *Advances in Neural Information Processing Systems*, 4, 1991.

[10] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Simulated annealing. *Journal of Chemical Physics*, 21:1087, 1953.

[11] A. Krogh and J.A. Hertz. A simple weight decay can improve generalisation. *Advances in Neural Information Processing Systems*, 4, 1992.

[12] S.J. Nowlan and G.E. Hinton. Simplifying neural networks by soft-weight sharing. *Neural Computation*, 4(4):473–493, 1992.

[13] S.E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. *Advances in Neural Information Processing Systems*, 2, 1990.

[14] M. Ferraro and T. Caelli. The relationship between integral transform invariances and lie group theory. *Journal of the Optical Society of America A*, 5:738–742, 1988.

[15] Christopher Zetzsche and Terry Caelli. Invariant pattern recognition using multiplr filter image represtations. *Computer Vision, Graphics, and Image Processing*, 45:251–262, 1989.