Chapter 1

Neural Networks

The field of Neural Networks has arisen from diverse sources, ranging from the fascination of mankind with understanding and emulating the human brain, to broader issues of copying human abilities such as speech and the use of language, to the practical commercial, scientific, and engineering diciplines of pattern recognition, modelling, and prediction. The pursuit of Technology is a strong driving force for researchers, both in academia and industry, in many fields of science and engineering. In neural networks, the excitement of technological progress is supplemented by the evocative and sometimes sinister thrill of reproducing intelligence itself.

Linear discriminants were introduced by Fisher in 1936 [17], as a statistical procedure for classification. Here the space of attributes can be partitioned by a set of hyperplanes, each defined by a linear combination of the attribute variables, treated as if they were numerical values. A similar model for logical processing was suggested by McCulloch and Pitts in 1943 [38] as a possible structure bearing similarities to neurons in the human brain, and they demonstrated that the model could be used to build any finite logical expression. The McCulloch-Pitts neuron consists of a weighted sum of its inputs, followed by a non-linear function called the em activation function, originally a threshold function. Formally,

$$y_j = \begin{cases} 1 \text{ if } \sum_i w_{ji} y_i - \mu_j \geq 0 \\ \\ 0 \text{ otherwise} \end{cases} \tag{1.1}$$

Other neuron models are quite widely used, for example in Radial Basis Function networks, which are discussed in detail in section 1.1.3.

Networks of McCulloch-Pitts neurons for arbitrary logical expressions were hand-crafted, until the ability to learn by reinforcement of behaviour

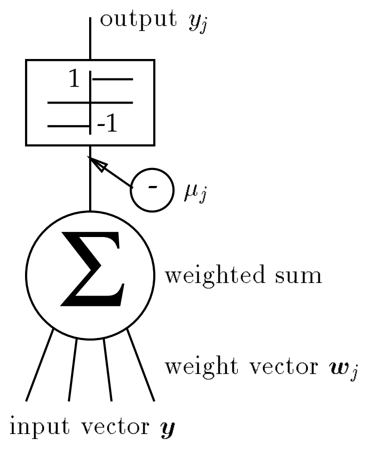
output y_j
1
-1
-
μ_j
weighted sum
weight vector $\boldsymbol{w}_j$
input vector $\boldsymbol{y}$

Figure 1.1: McCulloch and Pitts neuron

was developed in Hebb's book 'The Organisation of Behaviour' (Hebb, 1949)[25]. It was established that the functionality of neural networks was determined by the strengths of the connections between neurons; Hebb's learning rule prescribes that if the network responds in a desirable way to a given input, then the weights should be adjusted to increase the probability of a similar response to similar inputs in the future. Conversely, if the network responds undesirably to an input, the weights should be adjusted to decrease the probability of a similar response.

A distinction is often made, in pattern recognition, between *supervised* and *unsupervised* learning. The former describes the case where the the training data, measurements on the surroundings, are accompanied by labels indicating the class of event that the measurements represent, or more generally a desired response to the measurements. This is the more usual case in classification tasks, such as those forming the empirical basis of this book. The supervised learning networks described later in this chapter are the Perceptron and Multi Layer Perceptron (MLP), the Cascade Correlation learning architecture, and Radial Basis Function networks.

Unsupervised learning refers to the case where measurements are not accompanied by class labels. Networks exist which can model the structure of samples in the measurement, or attribute space, usually in terms of a probability density function, or by representing the data in terms of cluster centres and widths. Such models include Gaussian mixture models and

Kohonen networks.

Once a model has been made, it can be used as a classifier in one of two ways. The first is to determine which class of pattern in the training data each node or neuron in the model responds most strongly to, most frequently. Unseen data can then be classified according to the class label of the neuron with the strongest activation for each pattern. Alternatively, the Kohonen network or mixture model can be used as the first layer of a Radial Basis Function network, with a subsequent layer of weights used to calculate a set of class probabilities. The weights in this layer are calculated by a linear one-shot learning algorithm (See section 1.1.3.), giving radial basis functions a speed advantage over non-linear training algorithms such as most of the supervised learning methods. The first layer of a Radial Basis Function network can alternatively be initialised by choosing a subset of the training data points to use as centres.

1.1 Supervised Networks for Classification

In supervised learning, we have an instance of data, p, comprising an attribute vector $\mathbf{Y}_p^{(I)}$ and a target vector $\mathbf{Y}_p^{(T)}$. We process $\mathbf{Y}_p^{(I)}$ with a network, to produce an output $\mathbf{y}_p^{(T)}$, which has the same form as the target vector $\mathbf{Y}_p^{(T)}$.

The parameters of the network $\mathbf{w}$ are modified to optimise the match between outputs and targets, typically by minimising the total squared error $E = \frac{1}{2}\sum_p(\mathbf{y}_p^{(T)} - \mathbf{Y}_p^{(T)})^2$.

1.1.1 Perceptrons and Multi Layer Perceptrons

The activation of the McCulloch-Pitts neuron has been generalised to the form

$$y_j^{(T)} = f_j\left(\sum_i w_{ji} Y_i^{(I)}\right) \tag{1.2}$$

where the activation function, f_j can be any non-linear function. The nodes have been divided into an *input layer I* and an *output layer O*. The threshold level, or bias of equation (1.1) has been included in the sum, with the assumption of an extra component in the vector $\mathbf{Y}^{(I)}$ whose value is fixed at 1. Rosenblatt studied the capabilities of groups of neurons in a single layer, and hence all acting on the same input vectors; this structure was termed the Perceptron (Rosenblatt, 1958), [54] and Rosenblatt proposed the Perceptron Learning Rule for learning suitable weights for classification problems (Rosenblatt, 1962) [55]. When f is a hard threshold function, equation 1.2 defines a non-linear function across a hyperplane in the attribute space; with a threshold activation function the neuron output is simply 1 on one side of the hyperplane and 0 on the other. When combined in a perceptron structure, neurons can segment the attribute space into regions, and this forms the basis of the capability of perceptron networks to perform classification.

Minsky and Papert pointed out, however, that many real world problems do not fall into this simple framework, citing the exclusive-or problem as the simplest example. Here it is necessary to isolate two convex regions, joining them together in a single class. They showed that while this was not possible with a perceptron network, it can be done with a two layer perceptron structure (Minsky and Papert, 1969) [40]. This formed the Multi Layer Perceptron (MLP) which is widely in use today, although the Perceptron Learning Rule (also called the Delta Rule) could not be generalised to find weights for this structure.

A learning rule was proposed in 1985 which allows the multi layer perceptron to learn. This *Generalised Delta Rule* (section 1.1.2) defines a notion of back-propagation of error derivatives through the network (Werbos, 1974, Rumelhart, 1985 and 1986) [62, 9, 10], and enables a large class of models with different connection structures, or *architectures* to be trained. These publications initiated the recent academic interest in neural networks, and the field subsequently came to the attention of industrial users. This has resulted in a large number of academic publications and successful industrial applications.

1.1.2 Multi Layer Perceptron structure and functionality

Figure 1.2 shows the structure of a standard two-layer perceptron. The inputs form the *input nodes* of the network; the outputs are taken from the *output nodes*. The middle layer of nodes, visible to neither the inputs nor the outputs, is termed the *hidden layer*, and unlike the input and output layers, its size is not fixed. The hidden layer is generally used to make a *bottleneck*, forcing the network to make a simple model of the system generating the data, with the ability to generalise to previously unseen patterns.

The operation of this network is specified by

$$y_i^{(H)} = f^{(H)}\left(\sum_j w_{ij}^{(HI)} y_j^{(I)}\right)$$

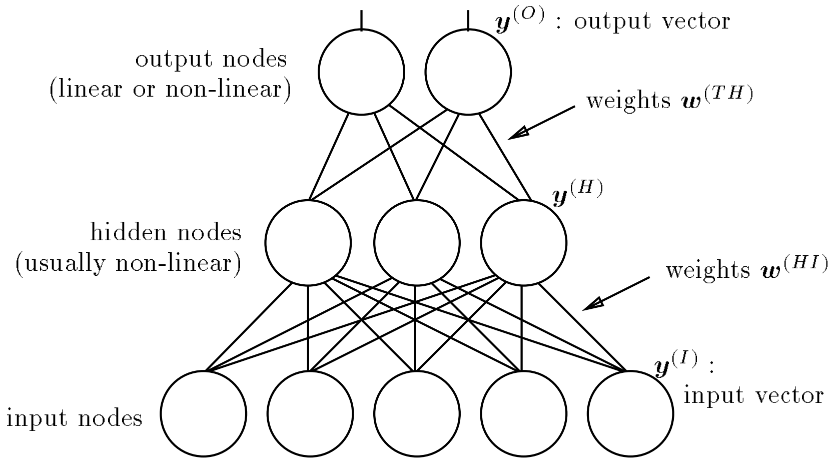

Figure 1.2: MLP Structure

$$y_i^{(T)} = f^{(T)}\left(\sum_j w_{ij}^{(OH)} y_j^{(H)}\right) \tag{1.3}$$

This specifies how input pattern vector $y^{(I)}$ is mapped into output pattern vector $y^{(0)}$, via the hidden pattern vector $y^{(H)}$, in a manner parameterised by the two layers of weights $\mathbf{w}^{(HI)}$ and $\mathbf{w}^{(TH)}$. The univariate functions $f^{(\cdot)}$ are typically each set to

$$f(x) = \frac{1}{1+e^{-x}} \tag{1.4}$$

which varies smoothly from 0 at $-\infty$ to 1 at ∞, as a threshold function would do abruptly.

If the number of hidden layer nodes is less than the number of degress of freedom inherent in the training data, the activations of the hidden nodes tend to form an orthogonal set of variables, either linear or non-linear combinations of the attribute variables, which span as large a subspace of the problem as possible. With a little extra constraint on the network, these internal variables form a linear or non-linear principal component representation of the attribute space. If the data has noise added that is not an inherent part of the generating system, then the principal component network acts as a filter of the lower-variance noise signal, provided the signal to noise ratio of the data is sufficiently high. This property gives MLPs the ability to generalise to previously unseen patterns, by modelling only the important underlying structure of the generating system. The hidden nodes can be regarded as detectors of abstract features of the attribute space.

Universal Approximators and Universal Computers

In the Multilayer Perceptron (MLP) such as the two-layer version in equation (1.3), the output-layer node values $\mathbf{y}^{(T)}$ are functions of the input-layer node values $\mathbf{Y}^{(I)}$ (and the weights $\mathbf{w}$). It can be shown (Funahashi, 1989) [21] that the two-layer MLP can approximate an arbitrary continuous mapping arbitrarily closely if there is no limit to the number of hidden nodes. In this sense the MLP is a universal function approximator. This theorem does not imply that more complex MLP architectures are pointless; it can be more efficient (in terms of the number of nodes and weights required) to use different numbers of layers for different problems. Unfortunately there is a shortage of rigorous principles on which to base a choice of architecture, but many heuristic principles have been invented and explored. Prominent among these are *symmetry principles* (Lang, 1990, Le Cun, 1989) [32, 33] and *constructive algorithms* (Wynne-Jones, 1991) [65].

The MLP is a *feedforward* network, meaning that the output vector $\mathbf{y}^{(T)}$ is a function of the input vector $\mathbf{Y}^{(I)}$ and some parameters $\mathbf{w}$; we can say

$$\mathbf{y}^{(T)} = F(\mathbf{y}^{(I)}; \mathbf{w}) \tag{1.5}$$

for some vector function F given in detail by (1.3) in the 2-layer case. It is also possible to define a recurrent network by feeding the outputs back to the inputs. The general form of a recurrent perceptron is

$$y_i(t+1) = f\left(\sum_j w_{ij} y_j(t)\right) \tag{1.6}$$

which could be written

$$\mathbf{y}(t+1) = F(\mathbf{y}(t); \mathbf{w}) \tag{1.7}$$

This is a discrete-time model; continuous-time models governed by a differential equation of similar structure are also studied.

Recurrent networks are universal computers in the sense that given an infinite number of nodes, they can emulate any calculation which can be done on a Turing machine. (The infinite number of nodes is needed to simulate the infinite Turing tape.) This result is easily proved for hard-threshold recurrent perceptrons by sketching a 1-node network which performs not-AND and another which functions as a FLIP-FLOP. These two elements are all that are required to build a computer.

This chapter focuses on feedforward neural network models because they are simpler to use, better understood, and closely connected with statistical classification methods. However recurrent networks attract a great deal of research interest because of their potential to serve as a vehicle for bringing statistical methods to bear on algorithm design (Rohwer 1991a, 1991b, 1992a, 1992b, Shastri & Ajjanagadde 1993) [48, 51, 52, 58, 47].

Training MLPs by nonlinear regression

In neural network parlance, *training* is the process of fitting a networks parameters (its weights) to given data. The training data consists of a set of examples of corresponding inputs and desired outputs, or "targets". Let the $p^{\underline{\text{th}}}$ example be given by input $Y_{ip}^{(I)}$ for input dimension i and target $Y_{ip}^{(T)}$ for target dimension i. Usually a least-squares fit is obtained by finding the parameters which minimize the *error measure*

$$E = \tfrac{1}{2}\sum_p \sum_i (y_{ip}^{(T)} - Y_{ip}^{(T)})^2. \tag{1.8}$$

where $y_{ip}^{(T)}$ are the output values obtained by substituting the inputs $Y_{ip}^{(I)}$ for $y_i^{(I)}$ in (1.3). If the fit is perfect, $E = 0$; otherwise $E > 0$.

Probabilistic interpretation of MLP outputs

If there is a one-to-many relationship between the inputs and targets in the training data, then it is not possible for any mapping of the form (1.5) to perform perfectly. It is straightforward to show (Bourlard & Wellekens, 1990) [6] that if a probability density $P(\mathbf{Y}^{(T)}|\mathbf{Y}^{(I)})$ describes the data, then the minimum of (1.8) is attained by the map taking $\mathbf{Y}^{(I)}$ to the average target

$$\int d\mathbf{Y}^{(T)} P(\mathbf{Y}^{(T)}|\mathbf{Y}^{(I)})\mathbf{Y}^{(T)}. \tag{1.9}$$

Any given network might or not be able to approximate this mapping well, but when trained as well as possible it will form its best possible approximation to this mean. Many commonly-used error measures in addition to (1.8) share this property (Hampshire & Pearlmuter, 1990) [23].

Usually classification problems are represented using *one-out-of-N* output coding. One output node is allocated for each class, and the target vector $\mathbf{Y}_p^{(T)}$ for example p is all 0's except for a 1 on the node indicating the correct class. In this case, the value computed by the $i^{\underline{\text{th}}}$ target node can be directly interpreted as the probability that the input pattern belongs to class i. Collectively the outputs express $P(\mathbf{Y}^{(T)}|\mathbf{Y}^{(I)})$. This not only provides helpful insight, but also provides a principle with which neural network models can be combined with other probabilistic models (Bourlard & Wellekens, 1990) [6].

The probabilistic interpretation of the the ouput nodes leads to a natural error measure for classification problems. Given that the value $y_{ip}^{(T)}$ output by the $i^{\underline{\text{th}}}$ target node given the $p^{\underline{\text{th}}}$ training input $\mathbf{Y}_p^{(I)}$, is $P(Y_{ip}^{(T)} = 1)$, so $1 - y_{ip}$ is $P(Y_{ip}^{(T)} = 0)$, the probability of the entire collection of training outputs $\mathbf{Y}^{(T)}$ is

$$P(\mathbf{Y}^{(T)}) = \prod_{ip} y_{ip}^{(T)\,Y_{ip}^{(T)}} \left(1 - y_{ip}^{(T)}\right)^{1-Y_{ip}^{(T)}} \tag{1.10}$$

This is the exponential of the *cross-entropy*,

$$E = \sum_p \sum_i \left(Y_{ip}^{(T)} \log y_{ip}^{(T)} + (1 - Y_{ip}^{(T)}) \log(1 - y_{ip}^{(T)}) \right) \tag{1.11}$$

Therefore the cross-entropy can be used as an error measure instead of a sum of squares (1.8). It happens that its minimum also lies at the average target (1.9), so the network outputs can still be interpreted probabilisitcally, and furthermore the minimisation of cross-entropy is equivalent to maximisation of the liklihood of the training data in classification problems. [1]

The probabilistic interpretation of MLP outputs in classification problems must be made with some caution. It only applies if the network is trained to its minimum error, and then only if the training data accurately represents the underlying probability density $P(\mathbf{Y}^{(T)}|\mathbf{Y}^{(I)})$. The latter condition is problematic if $\mathbf{Y}^{(I)}$ belongs to a continuous space or a large discrete set, because technically a large or infinite amount of data is required. This problem is intimately related to the overtraining and generalisation issues discussed below.

For the theoretical reasons given here, the cross-entropy is the most appropriate error measure for use in classification problems, although practical experience suggests it makes little difference. The sum of squares was used in the numerical experments reported in this book.

[1] The cross-entropy (1.11) has this interpretation when an input can simultaneously be a member of any number of classes, and membership of one class provides no information about membership of another. If an input can belong to one and only one class, then the simple entropy, obtained by dropping the terms involving $(1 - y)$, should be used.

Minimisation methods

Neural network models are trained by adjusting their weight matrix parameters $\mathbf{w}$ so as to minimize an error measure such as (1.8). In the simplest cases the netork outputs are linear in the weights, making (1.8) quadratic. Then the minimal error can be found by solving a linear system of equations. This special case is discussed in section 1.1.3 in the context of Radial Basis Function networks, which have this property. In the generic, non-linear case the minimisation is accomplished using a variant of *Gradient Descent*. This produces a *local* minimum, a $\mathbf{w}$ from which any infinitesimal change increases E, but not necessarily the *global* minimum of $E(\mathbf{w})$.

First order gradient based methods

The *gradient* $\nabla E(\mathbf{w})$ of $E(\mathbf{w})$ is the vector field of derivatives of E: $\nabla E(\mathbf{w}) = (\frac{dE(\mathbf{w})}{dw_1}, \frac{dE(\mathbf{w})}{dw_2}, ...)$ (a *field* because the vector depends on $\mathbf{w}$). A linear approximation to $E(\mathbf{w})$ in the infinitesimal vicinity of an arbitrary weight matrix $\mathbf{w}^0$ is given by

$$E(\mathbf{w}) = E(\mathbf{w}^0) + \nabla E(\mathbf{w}^0) \cdot (\mathbf{w} - \mathbf{w}^0) \tag{1.12}$$

Clearly then, at any point $\mathbf{w}$ of the parameter space (weight space) of the network, the vector ∇E points in the direction of fastest increase of E; *ie.*, of all the infinitesimal changes $\delta\mathbf{w}$ (of a given magnitude) which one could make to $\mathbf{w}$, a change in the direction of ∇E increases E the most. Consequently an adjustment of $\mathbf{w}$ in the direction of $-\nabla E$ provides the maximum possible decrease in E. The basic strategy in gradient descent is to compute the gradient and adjust the weights in the opposite direction.

The problem with this method is that the theorem on maximal descent only applies to infinitesimal adjustments. The gradient changes as well as the error, so the optimal direction for (infinetesimal) descent changes when $\mathbf{w}$ is adjusted. The *Pure* Gradient Descent algorithm requires a *step size* parameter η, chosen small enough for $\eta\nabla E$ to be effectively infinitesimal so far as obtaining descent is concerned, but otherwise as large as possible, in the interests of speed. The weights are repeatedly adjusted by

$$\mathbf{w} \leftarrow \mathbf{w} - \eta\nabla E(\mathbf{w}) \tag{1.13}$$

until the error E fails to descend.

In practice, trial and error is used to look for the largest step size η which will work. With large step sizes, the gradient will tend to change dramatically with each step. A popular heuristic is to us a moving average of the gradient vector in order find a systematic tendency. This is accomplished by adding a *momentum* term to (1.13), involving a parameter $\alpha \lesssim 1$:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E(\mathbf{w}) + \alpha \delta \mathbf{w}_{\mathrm{old}}. \tag{1.14}$$

Here $\delta \mathbf{w}_{\mathrm{old}}$ refers to the most recent weight change.

These methods offer the benefit of simplicity, but their performance depends sensitively on the parameters η and α (Toolenaere, 1990) [60]. Different values seem to be appropriate for different problems, and for different stages of training in one problem. This circumstance has given rise to a plethora of heuristics for adaptive *variable step size* algorithms (Toolenaere, 1990, Silva & Almeida, 1990, Jacobs, 1988) [60, 59, 27].

Second-Order methods

The underlying difficulty in first order gradient based methods is that the linear approximation (1.12) ignores the curvature of $E(\mathbf{w})$. This can be redressed by extending (1.12) to the quadratic approximation,

$$E(\mathbf{w}) = E(\mathbf{w}^0) + \nabla E(\mathbf{w}^0) \cdot \delta \mathbf{w} + \delta \mathbf{w} \nabla \nabla E(\mathbf{w}^0) \delta \mathbf{w} \tag{1.15}$$

where $\nabla\nabla E$ is the matrix with components $\frac{d^2 E}{dw_i dw_j}$, called the inverse *Hessian* (or the Hessian, depending on conventions), and $\delta \mathbf{w} = \mathbf{w} - \mathbf{w}^0$. The change $\delta \mathbf{w} = -\frac{1}{2} H \nabla E$, where $H^{-1} = \nabla\nabla E$, brings $\mathbf{w}$ to a stationary point of this quadratic form. This may be a minimum, maximum, or saddle point. If it is a minimum, then a step in that direction seems a good idea; if not, then a positive or negative step (whichever has a negative projection on the gradient) in the *conjugate gradient* direction, $H\nabla E$, is at least not unreasonable. Therefore a large class of algorithms has been developed involving the conjugate gradient.

Most of these algorithms require explicit computation or estimation of the Hessian H. The number of components of H is roughly half the square of the number of components of $\mathbf{w}$, so for large networks involving many weights, such algorithms lead to impractical computer memory requirements. But one algorithm, generally called *the* conjugate gradient algorithm, or the *memoryless* conjugate gradient algorithm, does not. This algorithm maintains an estimate of the conjugate direction without directly representing H.

The conjugate gradient algorithm uses a sequence of *linesearches*, one-dimensional searches for the minimum of $E(\mathbf{w})$, starting from the most recent estimate of the minimum and searching for the minimum in the direction of the current estimate of the conjugate gradient. Linesearch algorithms are comparitively easy because the issue of direction choice reduces to a binary choice. But because the linesearch appears in the inner loop of the conjugate gradient algorithm, efficiency is important. Considerable effort therefore goes into it, to the extent that the linesearch is typically the most complicated module of a conjugate gradient implementation. Numerical round-off problems are another design consideration in linesearch implementations, because the conjugate gradient is often nearly orthogonal to the gradient, making the variation of $E(\mathbf{w})$ along the conjugate gradient especially small.

The update rule for the conjugate gradient direction $\mathbf{s}$ is

$$\mathbf{s} \leftarrow -\nabla E + \alpha \mathbf{s}_{\text{old}} \tag{1.16}$$

where

$$\alpha = \frac{(\nabla E - \nabla E_{\text{old}}) \cdot \nabla E}{\nabla E_{\text{old}} \cdot \nabla E_{\text{old}}}. \tag{1.17}$$

(This is the *Polak-Ribiere* variant; there are others.) Somewhat intricate proofs exist which show that if E were purely quadratic in $\mathbf{w}$, $\mathbf{s}$ were initialised to the gradient, and the linesearches were performed exactly, then $\mathbf{s}$ would converge on the congugate gradient and E would converge on its minimum after as many interations of (1.16) as there are components of $\mathbf{w}$. In practice good performance is often obtained on much more general functions using very imprecise linesearches. It is necessary to augment (1.17) with a rule to reset $\mathbf{s}$ to $-\nabla E$ whenever $\mathbf{s}$ becomes too nearly orthogonal to the gradient for progress to continue.

An implementation of the conjugate gradient algorithm will have several parameters controlling the details of the linesearch, and others which define exactly when to reset $\mathbf{s}$ to $-\nabla E$. But unlike the step size and momentum parameters of the simpler methods, the performance of the conjugate gradient method is relatively insensitive to its parameters if they are set within reasonable ranges. All algorithms are sensitive to process for selecting initial weights, and many other factors which remain to be carefully isolated.

The MLP results reported in this book were obtained with a Polak-Ribere conjugate gradient algorithm, employing a linesearch method based on a combined quadratic approximation and divide-and-conquer strategy (Fletcher, 1980) [18]. A precise specification of the algorithm is no simpler than an inspection of the C source code in which it is implemented; this is available at the time of writing by anonymous ftp from uk.ac.aston.cs (134.151.52.106). It is benchmarked with several other algorithms in (Rohwer, 1991) ([49]).

Gradient Calculations in MLPs

It remains to discuss the computation of the gradient $\nabla E(\mathbf{w})$ in the case of an MLP neural network model with an error measure such as (1.8). The calculation is conveniently organised as a *back propagation of error* (Rumelhart, *et. al.*, 1986, Rohwer & Renals, 1988) [56, 53]. For a network with a single layer of hidden nodes, this calculation proceeds by propagating node output values y forward from the input to output layers for each training example, and then propagating quantities δ related to the output errors backwards through a linearised version of the network. Products of δs and ys then give the gradient. In the case of a network with an input layer (I), a single hidden layer (H), and an output or target layer (T), the calcualtion is:

$$y_i^{(H)} = f^{(H)}\left(\sum_j w_{ij}^{(HI)} y_j^{(I)}\right) \tag{1.18}$$

$$y_i^{(T)} = f^{(T)}\left(\sum_j w_{ij}^{(TH)} y_j^{(H)}\right) \tag{1.19}$$

$$\delta_{ip}^{(T)} = (y_{ip}^{(T)} - Y_{ip}^{(T)}) \tag{1.20}$$

$$\delta_{ip}^{(H)} = \sum_j \delta_{jp}^{(T)} f_{jp}^{\prime(T)} w_{ji}^{(TH)} \tag{1.21}$$

$$dE/dw_{ij}^{(TH)} = \sum_p \delta_{ip}^{(T)} f_{ip}^{\prime(T)} y_{jp}^{(H)} \tag{1.22}$$

$$dE/dw_{ij}^{(HI)} = \sum_p \delta_{ip}^{(H)} f_{ip}^{\prime(H)} Y_{jp}^{(I)} \tag{1.23}$$

The index p is summed over training examples, while the is and js refer to nodes, and $f_{jp}^{\prime(\cdot)} = \frac{d}{dx} f(x)\Big|_{x=f^{-1}\left(y_{jp}^{(\cdot)}\right)}$. This network architecture was used in the work reported in this book.

Online vs. Batch

Note that both the error E (1.8) and the gradient ∇E (1.22, 1.23) are a sum over examples. These could be estimated by randomly selecting a subset of examples for includion in the sum. In the extreme, a single example might be used for each gradient estimate. This is a *Stochastic Gradient* method. If a similar strategy is used without random selection, but with the data taken in the order it comes, the method is an *Online* one. If a sum over all

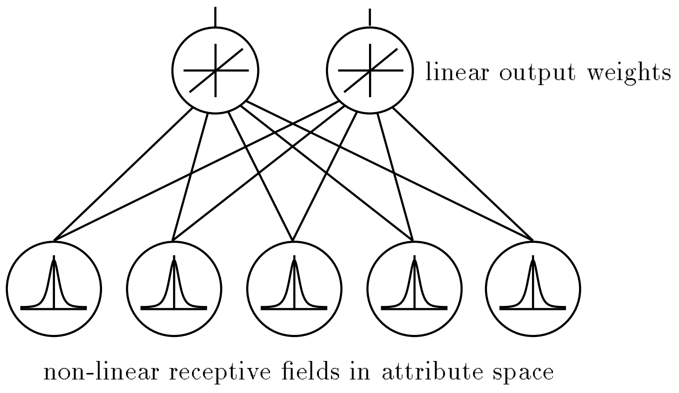

Figure 1.3: A Radial Basis Function Network

training data is performed for each gradient calculation, then the method is a *Batch* variety.

Online and Stochastic Gradient methods offer a considerable speed advantage if the approximation is servicable. For problems with large amounts of training data they are highly favoured. However, these approximations cannot be used directly in the conjugate gradient method, because it is built on procedures and theorems which assume that E is a given function of $\mathbf{w}$ which can be evaluated precisely so that meaningful comparisons can be made at nearby arguments. Therefore the stochastic gradient and Online methods tend to be used with simple step-size and momentum methods. There is some work on finding a compromise method (Moller 1993) [41].

1.1.3 Radial Basis Function Networks

The radial basis function network consists of a layer of units performing linear or non-linear functions of the attributes, followed by a layer of weighted connections to nodes whose outputs have the same form as the target vectors. This network offers a number of advantages over the multi layer perceptron under certain conditions, although the two models are computationally equivalent.

These advantages include a linear training rule once the locations in attribute space of the non-linear functions have been determined, and an underlying model involving localised functions in the attribute space, rather than the long-range functions occurring in perceptron-based models. The linear learning rule avoids problems associated with local minima; in particular it provides enhanced ability to make statments about the accuracy of the probabilistic interpretation of the outputs 1.1.2.

Figure 1.3 shows the structure of a radial basis function; the non-linearities comprise a position in attribute space at which the function is located (often referred to as the function's *centre*), and a non-linear function of the distance of an input point from that centre, which can be any function at all. Common choices include a gaussian response function, $\exp(-x^2)$ and inverse multiquadrics ($[z^2+c^2]^{-\frac{1}{2}}$), as well as non-local functions such as thin plate splines ($z^2 \log z$) and multiquadrics ($[z^2+c^2]^{\frac{1}{2}}$). Although it seems counter-intuitive to try and produce an interpolating function using non-localised functions, they are often found to have better interpolating properties *in the region populated by the training data.*

The Radial Basis Function network approach involves the expansion or pre-processing of input vectors into a high-dimensional space. This attempts to exploit a theorem of Cover in 1965 [8] which implies that a classification problem cast in a high-dimensional space is more likely to be linearly separable than would be the case in a low-dimensional space.

Training: choosing the centres and non-linearities

A number of methods can be used for choosing the centres for a radial basis function network. It is important that the distribution of centres in the attribute space should be similar to, or at least cover the same region as the training data. It is assumed that the training data is representative of the problem, otherwise good performance cannot be expected on future unseen patterns.

A first order technique for choosing centres is to take points on a square grid covering the region of attribute space covered by the training data. Alternatively, better performance might be expected if the centres were sampled at random from the training data itself, using some or all samples, since the more densely populated regions of the attribute space would have a higher resulution model than sparser regions. In this case, it is important to ensure that at least one sample from each class is used as a prototype centre. In the Statlog experiments, the number of samples required from each class was calculated before sampling, thereby ensuring this condition was met.

When centre positions are chosen for Radial Basis Function networks with localised non-linear functions such as Gaussian receptive fields, it is important to calculate suitable variances, or spreads for the functions. This ensures that large regions of space do not occur between centres, where no centres respond to patterns, and conversely, that no pair of centres respond nearly identically to all patterns. This problem is particularly prevalent in high dimensional attribute spaces because volume depends sensitively on radius. For a quantative discussion of this point, see (Prager & Fallside, 1989)[43]. In the Statlog experiments, the standard deviations of the Gaussian functions were set seperately for each coordinate direction to the distance to the nearest centre in that direction, multiplied by an arbitrary scaling parameter (set to 1.0).

Other methods include using a 'principled' clustering technique to position the centres, such as a Gaussian Mixture model or a Kohonen network. These models are discussed in section 1.2.

Training: optimising the weights

As mentioned in section (1.1.2), radial basis function networks are trained simply by solving a linear system. There are a few subtleties however, which are discussed here. Let $y_{jp}^{(H)}$ be the output of the $j^{\underline{\text{th}}}$ radial basis function on the $p^{\underline{\text{th}}}$ example. The output of each target node i is computed using the weights w_{ij} as

$$y_{ip}^{(T)} = \sum_j w_{ij} y_{jp}^{(H)} \tag{1.24}$$

Let the desired output for example p on target node i be $Y_{ip}^{(T)}$. The error measure (1.8) written out in full is then

$$E(\mathbf{w}) = \tfrac{1}{2} \sum_{ip} \left(\sum_j w_{ij} y_{jp}^{(H)} - Y_{ip}^{(T)} \right)^2 \tag{1.25}$$

which has its minimum where the derivative

$$\frac{dE}{dw_{kl}} = \sum_j \sum_p w_{kj} y_{jp}^{(H)} y_{lp}^{(H)} - \sum_p Y_{kp}^{(T)} y_{lp}^{(H)} \tag{1.26}$$

vanishes. Let $\mathbf{R}$ be the correlation matrix of the radial basis function outputs,

$$R_{ij} = \sum_p y_{jp}^{(H)} y_{ip}^{(H)}. \tag{1.27}$$

The weight matrix $\mathbf{w}^*$ which minimizes E lies where the gradient vanishes:

$$\mathbf{w}_{ij}^* = \sum_k \sum_p Y_{ip}^{(T)} y_{kp}^{(H)} \left(\mathbf{R}^{-1}\right)_{kj}. \tag{1.28}$$

Thus, the problem is solved by inverting the square $H \times H$ matrix $\mathbf{R}$, where H is the number of radial basis functions.

The matrix inversion can be accomplished by standard methods such as LU deomomposition (Renals & Rohwer, 1989) [46] (Press, *et. al.*) [44] if $\mathbf{R}$ is neither singular nor nearly so. This is typically the case, but things can go wrong. If two radial basis function centres are very close together a singular matrix will result, and a singular matrix is guaranteed if the number of training samples is not at least as great as H. There is no practical way to ensure a non-singular correlation matrix. Consequently the safest course of action is to us a slightly more computationally expensive singular value decomposition method. Such methods provide an approximate inverse by diagonalising the matrix, inverting only the eigenvalues which exceed zero by a parameter-specified margin, and transforming back to the original coordinates. This provides an optimal minimum-norm approximation to the inverse in the least-mean-squares sense.

Another approach to the entire problem is possible (Broomhead & Lowe, 1988) [7]. Let P be the number of training examples. Instead of solving the $H \times H$ linear system given by the derivatives of E (1.26), this method focuses on the linear system embedded in the error formula (1.25) itself:

$$\sum_j w_{ij} y_{jp}^{(H)} = Y_{ip}^{(T)}. \tag{1.29}$$

Unless $P = H$, this is a rectangular system. In general an exact solution does not exist, but the optimal solution in the least-squares sense is given by the pseudo-inverse (kohonen, 1989) [29] $\mathbf{y}^{(H)+}$ of $\mathbf{y}^{(H)}$, the matrix with elements $y_{ip}^{(H)}$:

$$\mathbf{w}^* = \mathbf{Y}^{(T)} \mathbf{y}^{(H)+} \tag{1.30}$$

This formula is applied directly. The identity $\mathbf{Y}^+ = \tilde{\mathbf{Y}}(\mathbf{Y}\tilde{\mathbf{Y}})^+$, where $\tilde{}$ denotes the matrix transpose, can be applied to (1.30) to show that the pseudo-inverse method gives the same result as (1.28):

$$\mathbf{w}^* = \mathbf{Y}^{(T)} \tilde{\mathbf{y}}^{(H)} \left(\mathbf{y}^{(H)} \tilde{\mathbf{y}}^{(H)} \right)^+ \tag{1.31}$$

The requirement to invert or pseudo-invert a matrix dependent on the entire dataset makes this a batch method. However an online variant is possible, known as *Kahlman Filtering* (Scalero & Tepedelenlioglu, 1992) [57]. It is based on the somewhat remarkable fact that an exact expression exists for updating the inverse correlation $\mathbf{R}^{-1}$ if another example is added to the sum (1.27), which does not require recomputation of the inverse.

1.1.4 Improving the Generalization of Feed-Forward Networks

Constructive Algorithms and pruning

A number of techniques have emerged recently, which attempt to improve on the perceptron and multilayer perceptron training algorithms by changing the architecture of the networks as training proceeds. These techniques include *pruning* useless nodes or weights, and *constructive algorithms* where extra nodes are added as required. The advantages include smaller networks, faster training times on serial computers, and increased generalization ability, with a consequent immunity to noise. In addition, it is frequently much easier to interpret what the trained netowork is doing. As was noted earlier, a minimalist networks uses its hidden layer to model as much of the problem as possible in the limited number of degrees of freedom available in its hidden layer. With such a network, one can then begin to draw analogies with other pattern classifiying techniques such as decision trees and expert systems.

To make a network with good generalization ability, we must determine a suitable number of hidden nodes. If there are too few, the network may not learn at all, while too many hidden nodes lead to over-learning of individual samples at the expense of forming a near optimal model of the data distributions underlying the training data. In this case, previously unseen patterns are labeled accoding to the nearest neighbour, rather than in accordance with a good model of the problem.

Early constructive algorithms such as *Upstart* (Frean, 1990) [20, 19] and the *Tiling Algorithm* (Mezard, 1989)[12] built multi-layer feed-forward networks of perceptron units (Rosenblatt, 1958)[54], which could be applied to problems involving binary input patterns. Convergence of such algorithms is guaranteed if the data is linearly separable, and use of the Pocket algorithm (Gallant, 1985)[22] for training allows an approximate solution to be found for non linearly-separable datasets. These networks do not usually include a stopping criterion to halt the creation of new layers or nodes, so every sample in the training data is learned. This has strong repercussions if the training set is incomplete, has noise, or is derived from a classification problem where the distributions overlap.

Later methods apply to more general problems and are suitable for statistical classification problems [2, 16, 24, 45, 66, 67]. They often build a single hidden layer, and incorporate stopping criteria which alow them to converge to solutions with good generalization ability for statistical problems. *Cascade Correlation* (Fahlman, 1990) [16] is an example of such a network algorithm, and is described in section 1.1.4.

Pruning has been carried out on networks in three ways. The first is a heuristic aproach based on identifying which nodes or weights contribute little to the mapping. After these have been removed, additional training leads to a better network than the original. An alternative technique is to include terms in the error function, so that weights tend to zero under certain circumstances. Zero weights can then be removed without degrading the network performance. This aproach is the basis of *regularization*, discussed in more detail in section 1.1.4. Finally, if we define the sensitivity of the global network error to the removal of a weight or node, we can remove the weights or nodes to which the global error is least sensitive. The sensitivity measure does not interfere with training, and involves only a small amount of extra computational effort. A full review of these tehniques can be found in (Wynne-Jones, 1991) [65].

Cascade Correlation: A Constructive Feed-Forward Network

Cascade Correlation is a paradigm for building a feed-forward network as training proceeds in a supervised mode (Fahlman, 1990) [16]. Instead of adjusting the weights in a fixed architecture, it begins with a small network, and adds new hidden nodes one by one, creating a multi-layer structure. Once a hidden node has been added to a network, its input-side weights are frozen and it becomes a permanent feature-detector in the network, available for output or for creating other, more complex feature detectors in later layers. Cascade correlation can offer reduced training time, and it determines the size and topology of networks automatically.

Cascade correlation combines two ideas: first the cascade architecture, in which hidden nodes are added one at a time, each using the outputs of all others in addition to the input nodes, and seond the maximisation of the correlation between a new unit's output and the residual classification error of the parent network. Each node added to the network may be of any kind. Examples include linear nodes which can be trained using linear algorithms, threshold nodes such as single perceptrons where simple learning rules such as the Delta rule or the Pocket Algorithm can be used, or non-linear nodes such as sigmoids or Gaussian functions requiring Delta rules or more advanced algorithms such as Fahlman's Quickprop. (Fahlman, 1988)[13, 14]

At each stage in training, each node in a pool of candidate nodes is trained on the residual error of the parent network. Of these nodes, the one whose output has the greatest correlation with the error of the parent is added permanently to the network. The error function minimised in this scheme is S, the sum over all output units of the magnitude of the correlation (or, more precisely, the covariance) between V, the candidate unit's value, and $E_{p,o}$, the residual error observed at output unit o for example p. S is defined by:

$$S = \sum_{o} \left| \sum_{p} (V_p - \overline{V})(E_{p,o} - \overline{E_o}) \right| . \tag{1.32}$$

The quantities $\overline{V}$ and $\overline{E_o}$ are the values of V and E_o averaged over all patterns.

In order to maximise S, the partital derivative of the error is calculated with respect to each of the weights coming into the node, w_i. Thus:

$$\frac{\partial S}{\partial w_i} = \sum_{p,o} \sigma_o (E_{p,o} - \overline{E_o}) f'_p I_{i,p} \tag{1.33}$$

where σ_o is the sign of the correlation between the candidate's value and the output o, f'_p is the derivative for pattern p of the candidate unit's activation function withe respect to the sum of its inputs, and $I_{i,p}$ is the input the candidate unit receives for pattern p.

The partial derivatives are used to perform gradient ascent to maximise S. When S no longer improves in training for any of the candidate nodes, the best candidate is added to the network, and the others are scrapped.

In benchmarks on a toy problem involving classification of data points forming two interlocked spirals, cascade correlation is reported to be ten to one hundred times faster than conventional back-propagation of error derivatives in a fixed architecture network. Empirical tests on a range of real problems (Yang, 1991) [68] indicate a speedup of one to two orders of magnitude with minimal degredation of classification accuracy. These results were only obtained after many experiments to determine suitable values for the many parameters which need to be set in the cascade correlation implementation. Cascade correlation can also be implemented in computers with limited precision (Fahlman, 1991) [15], and in recurrent networks (Hoehfeld, 1991) [26].

Bayesian Regularization

In recent years the formalism of Bayesian probability theory has been applied to the treatment of feedforward neural network models as nonlinear regression problems. This has brought about a greatly improved understanding of the generalisation problem, and some new techniques to improve generalisation. None of these techniques were used in the numerical experiments described in this book, but a short introduction to this subject is provided here.

An reasonable scenerio for a Bayesian treatment of feedforward neural networks is to presume that each target training data vector $\mathbf{Y}^{(T)}$ was produced by running the corresponding input training vector $\mathbf{Y}^{(I)}$ through some network and corrupting the output with noise from a stationary source. The network involved is assumed to have been drawn from a probability distribution $P(\mathbf{w})$, which is to be estimated. The most probable $\mathbf{w}$ in this distribution can be used as the optimal classifier, or a more sophisticated average over $P(\mathbf{w})$ can be used. (The latter technique is *marginalisation* (Mackay, 1992)[36].)

The notation used here for probability densities is somewhat cavalier. In discussions involving several probability density functions, the notation should distinguish one density function from another, and further notation should be used when such a density is indicated at a particular point; *eg.*, $P_{\mathbf{W}}$ can designate the density function over weights, and $P_{\mathbf{W}}(\mathbf{w})$ would designate this density at the particular point $\mathbf{w}$, which confusingly and unsignificantly has the same name as the label index of P. However, a tempting opportunity to choose names which introduce this confusion will arise in almost every instance that a density function is mentioned, so we shall not only succomb to the temptation, but furthermore adopt the common practice of writing $P(\mathbf{w})$ when $P_{\mathbf{W}}(\mathbf{w})$ is meant, in order to be concise. Technically, this is an appalling case of using a function argument name (which is ordinarily arbitrary) to designate the function.

The Bayesian analysis is built on a probabilistic interpretation of the error measure used in training. Typically, as in equations (1.8) or (1.11), it is additive over input-output pairs $\mathbf{Y}$; *ie.* it can be expressed as

$$E(\mathbf{Y};\mathbf{w}) = \sum_p e(\mathbf{Y}_p;\mathbf{w}) \tag{1.34}$$

for some function e, where $\mathbf{Y}$ is all the training data, the set of input-output pairs in the sum. $\mathbf{Y}$ is composed of all the input data $\mathbf{Y}^{(I)}$, regarded as fixed, and all the target data $\mathbf{Y}^{(T)}$, regarded as a noise-corrupted, $\mathbf{w}$-dependent function of $\mathbf{Y}^{(I)}$, drawn from a distribution with density function $P(\mathbf{Y}^{(T)}|\mathbf{w})$ (or technically $P(\mathbf{Y}^{(T)}|\mathbf{w},\mathbf{Y}^{(I)})$. The Bayesian argument requires the assumption that $P(\mathbf{Y}^{(T)}|\mathbf{w})$ is a function of E alone. Thus, different choices of E correspond to different probabilistic interpretations. Given this assumption, and the assumption that training data samples are produced independently of eachother,

$$P(\{\mathbf{Y}_1,\mathbf{Y}_2\}|\mathbf{w}) = P(\mathbf{Y}_1;\mathbf{w})P(\mathbf{Y}_2;\mathbf{w}) \tag{1.35}$$

the relationship between $E(\mathbf{Y};\mathbf{w})$ and $P(\mathbf{Y}|\mathbf{w})$ can only have the form

$$P(\mathbf{Y}|\mathbf{w}) = \frac{1}{Z_Y}e^{-\beta E(\mathbf{Y};\mathbf{w})} \tag{1.36}$$

for some parameter β. Z_Y is the normalisation term

$$Z_Y = \int d\mathbf{Y}^{(T)} e^{-\beta E(\mathbf{Y};\mathbf{w})}, \tag{1.37}$$

an integral over all possible target training data sets of the size under consideration.

If e in (1.34) is a function only of $\mathbf{y}_p^{(T)} - \mathbf{Y}_p^{(T)}$, as is (1.8), then Z_Y turns out to be independant of $\mathbf{w}$ [2], a result which is useful later. The only common form of e which does not have this form is the cross-entropy (1.11). But this is normally used in classification problems, in which case (1.11) and (1.10) together justify the assumption that $P(\mathbf{Y}^{(T)}|\mathbf{w})$ depends only on $E(\mathbf{Y};\mathbf{w})$ and imply for (1.37) that $\beta = 1$ and $Z_Y = 1$, so Z_Y is still independent of $\mathbf{w}$.

Density (1.36) can also be derived from somewhat different assumptions using a maximum-entropy argument (Bilbro & Van den Bout, 1992)[3]. It plays a prominent role in thermodynamics, and thermodynamics jargon has drifted into the neural networks literature partly in consequence of the analogies it underlies.

The probability of the weights given the data $P(\mathbf{w}|\mathbf{Y}^{(T)})$ is of greater interest than the probability of the data given the weights $P(\mathbf{Y}^{(T)}|\mathbf{w})$ (the *likelihood*), but unfortunately the additivity argument does not go through for this. Instead, Bayes' rule

$$P(\mathbf{w}|\mathbf{Y}^{(T)}) = \frac{P(\mathbf{Y}^{(T)}|\mathbf{w})P^0(\mathbf{w})}{P(\mathbf{Y}^{(T)})}. \tag{1.38}$$

can be used to convert $P(\mathbf{Y}^{(T)}|\mathbf{w})$ from equation (1.36), and a prior over the weights $P^0(\mathbf{w})$, into the desired distribution. The probability of the data $P(\mathbf{Y}^{(T)})$ is given by the normalisation condition as

$$P(\mathbf{Y}^{(T)}) = \int d\mathbf{w} P(\mathbf{Y}^{(T)}|\mathbf{w})P^0(\mathbf{w}). \tag{1.39}$$

Bayesian methods inevitably require a prior, $P^0(\mathbf{w})$ in this case. $P^0(\mathbf{w})$ must express the notion that some weight matrices are more reasonable, *a priori*, than others. As discussed above, this is normally expressed through regularisation terms added to the error measure. For example, the view that large weights are unreasonable might be expressed by adding a "weight decay" term of the form $\alpha\mathbf{w}\cdot\mathbf{w}$ to $E(\mathbf{Y};\mathbf{w})$.

[2] There is a further technicality; the integral (1.37) over target data must be with respect to uniform measure, which may not always be reasonable.

Typically, the regularisation error $\alpha E(\mathbf{w})$ is additive over the weights and an independance assumption like (1.35) is reasonable, so given that the prior depends only on the regularisation term, the it has the form

$$P^0(\mathbf{w}) = \frac{1}{Z^0} e^{-\alpha E(\mathbf{w})} \tag{1.40}$$

where Z^0 is given by normalisation.

Assembling all the pieces, the posterior probability of the weights given the data is

$$P(\mathbf{w}|\mathbf{Y}^{(T)}) = \frac{e^{-\beta E\left(\mathbf{Y}^{(T)};\mathbf{w}\right)-\alpha E(\mathbf{w})}}{\int d\mathbf{w}' e^{-\beta E\left(\mathbf{Y}^{(T)};\mathbf{w}'\right)-\alpha E(\mathbf{w}')}} \tag{1.41}$$

provided that (1.37) does not depend on $\mathbf{w}$. This ensures that the denominator of (1.41) does not depend on $\mathbf{w}$, so the usual training process of minimising $E\left(\mathbf{Y}^{(T)};\mathbf{w}\right) + \frac{\alpha}{\beta}E(\mathbf{w})$ finds the maximum of $P(\mathbf{w}|\mathbf{Y}^{(T)})$.

The Bayesian method helps with one of the most troublesome steps in the regularisation approach to obtaining good generalisation, deciding the values of the regularisation parameters. The ratio α/β expresses the relative importance of smoothing and data-fitting, which deserves to be decided in a principled manner. The Bayesian *Evidence* formalism provides a principle and an implementation. It can be computationally demanding if used precisely, but there are practicable approximations.

The Evidence formalism simply assumes a prior distribution over the regularisation parameters, and sharpens it using Bayes' rule:

$$P(\alpha,\beta|\mathbf{Y}^{(T)}) = \frac{P(\mathbf{Y}^{(T)}|\alpha,\beta)P^0(\alpha,\beta)}{P(\mathbf{Y}^{(T)})}. \tag{1.42}$$

If a uniform prior $P^0(\alpha,\beta)$ is assumed, then the most likely regularisation parameters are those which maximise the *evidence* $P(\mathbf{Y}^{(T)}|\alpha,\beta)$, which is given by (1.39), the denominator of (1.38). Note with reference to (1.38) that the goal of maximising the evidence opposes the goal of maximising $P(\mathbf{w}|\mathbf{Y}^{(T)})$; the regularisation parameters α and β, and the weights $\mathbf{w}$ are optimised for opposing purposes. This expresses the Bayesian quantification of the compromise between data fitting and smoothing.

This method of setting regularisation parameters does not provide a guarantee against overfitting (Wolpert, 1992) [63], but it helps. In setting the regularisation parameters by maximising (1.39) $P(\mathbf{Y}^{(T)})$, one attempts to find a prior $P^0(\mathbf{w})$ under which "usually" networks $\mathbf{w}$ fits the data $\mathbf{Y}$ well. This objective is not diametrically opposed to the later objective of selecting the best-fitting $\mathbf{w}$ Indeed, the distribution $P(\mathbf{w})$ which maximises the evidence is one which is concentrated on a single overfit $\mathbf{w}$. This is prevented only if the the distribution of weight matrices parameterised by the regularisation parameters does not include such highly concentrated distributions. Therefore it remains an art to select reasonable functional forms for the regularisers, but once selected, the determination of the parameters themselves is a matter of calculation. The art of selecting regularisation functions has become an interesting research area (Hinton & Nowlan, 1992) [42].

The calculation of (1.42) involves an integration which is generally non-trivial, but which can be done easily in a Gaussian approximation. Typically this is good enough. This requires computation of the second derivatives of the error measure, which is prohibative for large problems, but in this case a further approximation is possible and often adequate (Mackay, 1992) [37].

1.2 Unsupervised Learning

Interest in Unsupervised Learning has increased greatly in recent years. It offers the possibility of exploring the structure of data without guidance in the form of class information, and can often reveal features not previously expected or known about. These might include the division of data that was previously thought to be a single uniform cluster, into a number of smaller groups, each with separate identifiable properties. The clusters found offer a model of the data in terms of cluster centres, sizes and shapes, which can often be described using less information, and in fewer parameters than were required to store the entire training data set. This has obvious advantages for storing, coding, and transmitting stochastically generated data; if its distribution in the attribute space is known, equivalent data can be generated from the model when required.

While general, unsupervised learning methods such as Boltzmann machines are computationally expensive, iterative clustering algorithms such as Kohonen networks, K-means clustering and Gaussian Mixture models offer the same modelling power with greatly reduced training time. Indeed, while class labels are not used to constrain the structure learned by the models, freedom from this constraint coupled with careful initialisation of the models using any prior information available about the data, can yield very quick and effective models. These models, known collectively as *Vector Quantizers*, can be used as the non-linear part of supervised learning models. In this case a linear part is added and trained later to implement the mapping from activation in different parts of the model, to probable classes of event generating the data.

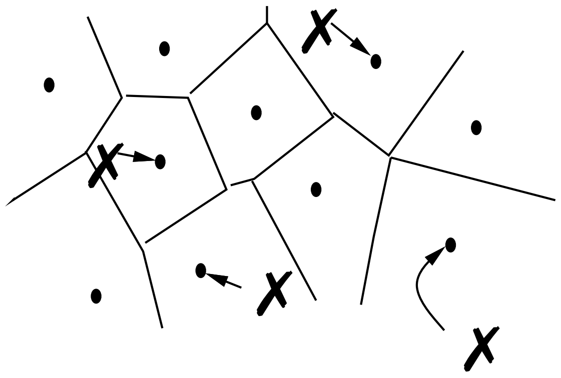

Figure 1.4: K Means clustering: within each patch the centre is moved to the mean position of the patterns

1.2.1 The K-means clustering algorithm

The principle of clustering requires a representation of a set of data to be found which offers a model of the distribution of samples in the attribute space. The K-means algorithm (eg. Krishnaiah & Kanal, 1982) [31] achieves this quickly and efficiently as a model with a fixed number of cluster centres, determined by the user in advance. The cluster centres are initially chosen from the data, and each centre forms the *code vector* for the patch of the input space in which all points are closer to that centre than to any other. This division of the space into patches is known as a *Voronoi tesselation.* Since the initial allocation of centres may not form a good model of the probability distribution function (PDF) of the input space, there follows a series of iterations where each cluster centre is moved to the mean position of all the training patterns in its tesselation region.

A generalised variant of the K-means algorithm is the Gaussian Mixture Model, or Adaptive K-means. In this scheme, Voronoi tesselations are replaced with soft transitions from one centre's receptive field to another's. This is achieved by assigning a variance to each centre, thereby defining a Gaussian kernel at each centre. These kernels are mixed together by a set of mixing weights to approximate the PDF of the input data, and an efficient algorithm exists to calculate iteratively a set of mixing weights, centres, and variances for the centres. (Dubes, 1976, and Wu, 1991) [11, 64] While the number of centres for these algortihms is fixed in advance in more popular implementations, some techniques are appearing which allow new centres to be added as training proceeds. (Wynne-Jones, 1992 and 1993) [66, 67]

1.2.2 Kohonen networks and Learning Vector Quantizers

Kohonen's network algorithm (Kohonen, 1984) [28] also provides a Voronoi tesselation of the input space into patches with corresponding code vectors. It has the additional feature that the centres are arranged in a low dimensional structure (usually a string, or a square grid), such that nearby points in the topological structure (the string or grid) map to nearby points in the attribute space. Structures of this kind are thought to occur in nature, for example in the mapping from the ear to the auditory cortex, and the retinotopic map from the retina to the visual cortex or optic tectum.

In training, the *winning node* of the network, which is the nearest node in the input space to a given training pattern, moves towards that training pattern, while dragging with its neighbouring nodes in the network topology. This leads to a smooth distribution of the network topology in a non-linear subspace of the training data.

Vector Quantisers that conserve topographic relations between centres are also particularly useful in communications, where noise added to the coded vectors may corrupt the representation a little; the topographic mapping ensures that a small change in code vector is decoded as a small change in attribute space, and hence a small change at the output. These models have been studied extensively, and recently unified under the framework of Bayes' theory. (Luttrell, 1990, 1993) [34, 35]

The Learning Vector Quantiser provides a *supervised* vector quantization, where network nodes have class labels associated with them. The Kohonen Learning Rule is used when the winning node represents the same class as a new training pattern, while a difference in class between the winning node and a training pattern causes the node to move *away* from the training pattern by the same distance. Learning Vector Quantisers are reported to give excellent performance in studies on statistical and speech data (Kohonen et al, 1988) [30].

1.2.3 RAMnets

One of the oldest practical neurally-inspired classification algorithms is still one of the best. It is the n-tuple recognition method introduced by Bledsoe and Browning [5, 4], which later formed the basis of a commercial product [1]. The algorithm is simple. The patterns to be classified are bit strings of a given length. Several (let us say N) sets of n bit locations are selected randomly. These are the n-tuples. The restriction of a pattern to an n-tuple can be regarded as an n-bit number which constitutes a 'feature' of the pattern. A pattern is classified as belonging to the class for which it has the most features in common with at least 1 pattern in the training data.

To be precise, the class assigned to unclassified pattern u is

$$\operatorname*{argmax}_{c}\left(\sum_{i=1}^{N}\Theta\left(\sum_{v\in\mathcal{C}_c}\delta_{\alpha_i(u),\alpha_i(v)}\right)\right) \tag{1.43}$$

where $\mathcal{C}_c$ is the set of training patterns in class c, $\Theta(x) = 0$ for $\Theta \leq 0$, $\Theta(x) = 1$ for $\Theta > 0$, $\delta_{i,j}$ is the Kronecker delta ($\delta_{i,j} = 1$ if $i = j$ and 0 otherwise.) and $\alpha_i(u)$ is the $i^{\underline{\text{th}}}$ feature of pattern u:

$$\alpha_i(u) = \sum_{j=0}^{n-1} u_{\eta_i(j)} 2^j. \tag{1.44}$$

Here u_i is the $i^{\underline{\text{th}}}$ bit of u and $\eta_i(j)$ is the $j^{\underline{\text{th}}}$ bit of the $i^{\underline{\text{th}}}$ n-tuple.

With C classes to distinguish, the system can be implemented as a set of NC RAMS, in which the memory content $m_{ci\alpha}$ at address α of the $i^{\underline{\text{th}}}$ RAM allocated to class c is

$$m_{ci\alpha} = \Theta\left(\sum_{v\in\mathcal{C}_c}\delta_{\alpha,\alpha_i(v)}\right). \tag{1.45}$$

Thus $m_{ci\alpha}$ is set if any pattern of $\mathcal{C}_c$ has feature α and unset otherwise. Recognition is accomplished by tallying the set bits in the RAMS of each class at the addresses given by the features of the unclassified pattern.

RAMnets are impressive in that they can be trained faster than MLPs or radial basis function networks by orders of magnitude, and often provide comparable results. Experimental comparisons between RAMnets and other methods can be found in [50].

1.3 DIPOL92

DIPOL92[3] is a learning algorithm which constructs an optimised piecewise linear classifier by a two step procedure. In the first step the initial positions of the discriminating hyperplanes are determined by pairwise linear regression. To optimise these positions in dependence on the misclassified patterns an error criterion function is defined. This function is then minimised by a gradient descent procedure for each hyperplane separately. As an option in the case of non–convex classes (e.g. if a class has a multimodal probability distribution) a clustering procedure decomposing the classes into appropriate subclasses can be applied. (In this case DIPOL92 is really a three step procedure.)

1.3.1 Pairwise Linear Regression

Regression is one of the most widely used methods of description of data relationships among independent variables $\vec{x} = (x_1, \ldots, x_m) \in X \subset R^m$ and a dependent variable b. For simplicity a linear regression function W , here with intercept w_0, is considered :

$$W : X \rightarrow R \tag{1.46}$$

with

$$W(\vec{x}) = w_0 + w_1 x_1 + \ldots + w_m x_m \tag{1.47}$$

and with the notation $\vec{w} = (w_1, \ldots, w_m)$.

The equations for the determination of the unknown coefficients of the regression function $w_0, w_1, \ldots, w_m$ are in matrix form

$$A(w_0, \vec{w})^T = (\vec{e}, M)(w_0, \vec{w})^T = \vec{b} \tag{1.48}$$

where $\vec{e}$ is a column of ones, the matrix $M = (x_{ik})$ contains in the row k the k-th vector of the dataset $\vec{x} = (x_1, \ldots, x_m)$ and the vector $\vec{b}$ contains the dependent variables b.

In general this system of linear equations is overdetermined, i.e., the number of equations n is greater than the number of unknown parameters (n$\gg$m). Systems of linear equations have a unique solution only if rank(A)=rank($A, \vec{b}$). Therefore the aim is to determine a solution of the system so that the vector of residuals

[3]Developed in the frame of the WISCON project supported by the German Ministry of Research and Technology, by Barbara Schulmeister and Fritz Wysotzki of Fraunhofer-Institut für Informations und Datenverarbeitung, Einrichtung für Prozeßoptimierung, Kurstraße 33, D-10117-Berlin, Germany

$$\vec{r} = \vec{b} - A(w_0, \vec{w})^T \tag{1.49}$$

is as small as possible. It is convenient to minimise the Euclidian length of $\vec{r}$

$$||\vec{r}||^2 = ||\vec{b} - A(w_0, \vec{w})^T||^2 \rightarrow Minimum \tag{1.50}$$

This minimum-squared error solution is found by the Cholesky method. If the matrix of the system of equations is singular, that means, the system cannot be solved uniquely, then a special solution is chosen from the linear solution manifold.

The linear regression can be used to the discrimination of two classes of patterns k_1 and k_2 by defining the dependent variable b in the following manner :

$$\text{if } \vec{x}\epsilon k_1, \text{ then } b = +1 \tag{1.51}$$
$$\text{if } \vec{x}\epsilon k_2, \text{ then } b = -1 \tag{1.52}$$
$$\tag{1.53}$$

All $\vec{y}\epsilon R^m$ with $W(\vec{y}) = 0$ define a hyperplane in R^m, the vector $\vec{w}$ is the normal vector of the hyperplane, the absolute value of $\frac{w_0}{||\vec{w}||}$ corresponds to the distance of the hyperplane from the origin of the coordinate system and for $\vec{x}\epsilon X$ the absolut value of $\frac{W(\vec{x})}{||\vec{w}||}$ corresponds to the distance of $\vec{x}$ from the hyperplane. Then a pattern $\vec{x}$ is correctly classified if

$$W(\vec{x}) > 0 \text{ for } \vec{x}\epsilon k_1, \tag{1.54}$$
$$W(\vec{x}) < 0 \text{ for } \vec{x}\epsilon k_2. \tag{1.55}$$

For each pair of classes a discriminating regression function is calculated.

1.3.2 Learning Procedure

The classification problem from the statistical point of view is the following. The given set of patterns of the classes is a random sample of the underlying populations of the classes. The aim of statistical classification is to discriminate these populations, for example by estimating the parameters of an assumed distribution from the given patterns. Here another view is held. The aim is to establish a classifier, which best discriminates the given classes of patterns. Therefore the initial positions of the discriminating hyperplanes obtained by linear regression are changed in a learning procedure. The following criterion function is defined. For all misclassified patterns the squared distances from the corresponding decision hyperplane multiplied with the costs for these misclassifications are summed. Supposing $W = 0$ defines the decision hyperplane between the classes k_1 and k_2, respectively. Let m_1 be the set of all misclassified patterns of class k_1, i.e., $x \epsilon k_1$ and $W(\vec{x}) < 0$, let m_2 be the set of all misclassified patterns of class k_2, i.e., $x \epsilon k_2$ and $W(\vec{x}) > 0$, and let $cost(k_i, k_j)$ be the costs of the misclassification of the class k_i into the class k_j:

$$F(W) = cost(k_1, k_2) * \sum_{\vec{x} \epsilon m_1} \frac{W(\vec{x})^2}{||\vec{x}||^2} + cost(k_2, k_1) * \sum_{\vec{x} \epsilon m_2} \frac{W(\vec{x})^2}{||\vec{x}||^2} \qquad (1.56)$$

This means that costs are included explicitly in the learning procedure which consists of minimizing the criterion function with respect to $w_0, w_1, \ldots, w_m$ by a gradient descent algorithm for each decision surface successively. The regression functions provide suitable starting positions of the discriminating hyperplanes for the iterative learning procedure.

1.3.3 Clustering of Classes

To handle problems with non-convex especially non simply-connected class regions, one can apply a clustering procedure before the linear regression is carried out. For solving the clustering problem a minimum squared error algorithm is used. From some initial partition of a class k into nk clusters k_i $(i = 1, \ldots, nk)$ with nk_i pattern and with mean vectors $\vec{s}_i$ $(i = 1, \ldots, nk)$

$$\vec{s}_i = \frac{1}{nk_i} \sum_{x \epsilon k_i} \vec{x} \qquad (1.57)$$

the criterion function

$$J = \sum_{i=1}^{nk} \sum_{\vec{x} \epsilon k_i} ||\vec{x} - \vec{s}_i||^2 \qquad (1.58)$$

is calculated. Patterns are moved from one cluster to another if such a move will improve the criterion function J. The mean vectors and the criterion function are updated after each pattern move. Like hill–climbing algorithms in general, these approaches guarantee local but not global optimisation. Different initial partitions and sequences of the training patterns can lead to different solutions. In the case of clustering the number of two-class problems increases correspondingly. It is to be noted that by the combination of the clustering algorithm with the regression technique the number and initial positions of discriminating hyperplanes are fixed apriori (i.e. before learning) in a reasonable manner, even in the case that some classes have multimodal distributions (i.e consist of several subclasses). Thus a well known bottleneck of Artificial Neural Nets can at least be partly avoided.

1.3.4 Description of the Classification Procedure

If the discriminating hyperplanes were calculated then any pattern $\vec{x} = (x_1, \ldots, x_m)$ (contained in the training set or not) can be classified, i.e. the class predicted. For the pairwise discrimination of the nc classes $ncc = nc(nc-1)/2$ hyperplanes W^i are calculated (in the case of clustering the number nc is changed into $nc + n_{clust}$). The following ncc-dimensional vector $\vec{V_k}$ is formed for each class k: if the function W^i discriminates the classes k_1 and k_2, then the ith component $V_{k,i}$ is equal to 1, if $k = k_1$, is equal to -1, if $k = k_2$, and is equal to 0 in all other cases. On the basis of the discriminant functions a vector function sw is defined for each pattern $\vec{x}$:

$$sw : X \rightarrow \{1, 0, -1\}^{ncc} \tag{1.59}$$

with

$$sw(\vec{x})_i = sign(W^i(\vec{x})). \tag{1.60}$$

For all classes k the scalar products

$$S_k(\vec{x}) : \{1, 0, -1\}^{ncc} \times \{1, 0, -1\}^{ncc} \rightarrow G \tag{1.61}$$

(G is the set of integers) are defined with

$$S_k(\vec{x}) = \sum_{i=1}^{ncc} V_{k,i} * sw(\vec{x})_i. \tag{1.62}$$

A pattern $\vec{x}$ is uniquely classified by the discriminating hyperplanes W^i $(i = 1, \ldots, ncc)$ into the class k if

$$S_k(\vec{x}) = nc - 1, \tag{1.63}$$

i.e., with respect to the $nc - 1$ hyperplanes, which discriminate the class k from the other $nc - 1$ classes, the pattern $\vec{x}$ is placed in the halfspace, belonging to class k ($V_{k,i}$ and $W^i(\vec{x})$ have the same sign for all $V_{k,i} \neq 0$). For all other classes j, $j \neq k$, $S_j < nc - 1$ is valid, because at least with respect to the hyperplane, which discriminates class j from class k the pattern $\vec{x}$ is placed in the halfspace of class k ($V_{j,i}$ and $W^i(\vec{x})$ have not the same sign). A pattern $\vec{x}$ is not uniquely classified if

$$\max_j S_j(\vec{x}) = mc < nc - 1. \tag{1.64}$$

In this case all classes j were determined with $S_j(\vec{x}) = mc$. If there is only one such class then $\vec{x}$ will be assigned to this class. If there are several classes let M be the set of the classes with this property, $M = \{j_1, \ldots, j_l\}$. For each class j_i all hyperplanes discriminating the class j_i against all other classes are found. Those of the hyperplanes $W_{j_i}^r$ for each class j_i are selected for which $\vec{x}$ is misclassified, i.e., for each class j_i a set of hyperplanes $H_{j_i} = \{W_{j_i}^1, \ldots, W_{j_i}^r\}$ is determined for which $\vec{x}$ is not in the halfspace of class j_i. The Euclidian distance of $\vec{x}$ to all these hyperplanes $W_{j_i}^s$ are calculated. $\vec{x}$ is assigned to that class for which the minimum

$$\min_{j_i \epsilon M} \min_{s=1,\ldots,r} W_{j_i}^s \tag{1.65}$$

is reached.

1.3.5 The Program DIPOL92

DIPOL92 was developed at the Branch Lab for Process Optimisation Berlin of the Fraunhofer-Institute for Information and Data Processing. (An algorithm based on similar principles was developed in the early seventies at the Academy of Sciences of the former GDR (Unger, 1981, Meyer-Brötz, 1970)[39, 61]. The program DIPOL92 is written in C++ and is easy to set up and run under UNIX normally in the batch mode. The input data of the program are feature vectors with real-valued components, logical and discrete data must be converted into numerical ones. The algorithm uses the standard StatLog attribute value data format. DIPOL92 does not have a mechanism to deal with unknown values. They must be replaced by some estimated values for example the mean of the values of the corresponding attribute. DIPOL92 accepts a cost matrix in the learning phase. The output of the algorithm is a confusion matrix, the error rate and the costs of misclassification. Optionally the output of the developed classifier (the coefficients of the discriminant functions and some additional information for later use) is possible. If the clustering option is used the number of clusters for the classes must be specified by the user. This number of clusters is the only parameter of the algorithm.

1.4 Comparison with other algorithms

Seen from a more general point of view DIPOL92 is a combination of a statistical part (regression) with a learning procedure typical for Artificial

Neural Networks. Compared with most neural network algorithms an advantage of DIPOL92 is the possibility to determine the number and initial positions of the discriminating hyperplanes (corresponding to neurons) *a priori*, i.e. before learning starts. Using the clustering procedure this is true even in the case that a class has several distinct subclasses. There are many relations and similarities between statistical and neuronal net algorithms but until now a systematic study of these relations is still lacking. Another distinguishing feature of DIPOL92 is the introduction of Boolean variables (signs of the normals of the discriminating hyperplanes) for the description of class regions on a symbolic level and using them in the decision procedure. This way additional layers of *hidden units* can be avoided.

Bibliography

[1] I. Aleksander, W. V. Thomas, and P. A. Bowden. Wisard: A radical step forward in image recognition. *Sensor Review*, 4:120–124, 1984.

[2] T. Ash. Dynamic node creation in back-propagation networks. ICS Report 8901, Institute of Cognitive Science, University of California, San Diego, La Jolla, California 92093, USA, 1989.

[3] G. Bilbro and D. Van den Bout. Maximum entropy and learning theory. *Neural Computation*, 4:839–853, 1992.

[4] W. W. Bledsoe. Further results on the n-tuple pattern recognition method. *IRE Trans. Comp.*, EC-10:96, 1961.

[5] W. W. Bledsoe and I. Browning. Pattern recognition and reading by machine. In *Proceedings of the Eastern Joint Computer Conference*, pages 232–255, Boston, 1959.

[6] H. Bourlard and C. J. Wellekens. Links between Markov models and multilayer perceptrons. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(12):1–12, 1990.

[7] D. S. Broomhead and David Lowe. Multi-variable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[8] T.M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, 14:326–334, 1965.

[9] G. E. Hinton D. E. Rumelhart and R. J. Williams. Learning internal representations by back-propagating errors. In D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, chapter 8. MIT Press, Cambridge, MA, USA, 1985.

[10] G. E. Hinton D. E. Rumelhart and R. J. Williams. Learning representatinos by back-propagating errors. *NAture*, 323:533–536, 1986.

[11] R. Dubes and A. K. Jain. Clustering techniques: The user's dilemma. *Pattern Recognition*, 8:247–260, 1976.

[12] M. ézard and J. P. Nadal. Learning in feed-forward layered networks: The tiling algorithm. *Journal of Physics A: Mathematics, General*, 22:2191–2203, 1989.

[13] Scott E. Fahlman. An empirical study of learning speed in back-propagation. Technical Report CMU-CS-88-162, Carnegie Mellon University, USA, June 1988.

[14] Scott E Fahlman. Faster learning variation on back-propagation: An empirical study. In *Proccedings of the 1988 Connectionist Models Summer School*. Morgan Kaufmann, 1988.

[15] Scott E. Fahlman. The recurrent cascade-correlation architecture. Technical Report CMU-CS-91-100, Carnegie Mellon University, USA, May 1991.

[16] Scott E. Fahlman and Christian Lebière. The cascade correlation learning architecture. In David S. Tourzetsky, editor, *Advances in Neural Information Processing Systems 2*, pages 524–532. Morgan Kaufmann, 1990.

[17] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.

[18] R. Fletcher. *Practical Methods of Optimization*. Wiley, 1980.

[19] Marcus Frean. *Short Paths and Small Nets: Optimizing Neural Computation*. PhD thesis, University of Edinburgh, UK, 1990.

[20] Marcus Frean. The upstart algorithm: A method for constructing and training feed-forward neural networks. *Neural Computation*, 2:198–209, Summer 1990.

[21] K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183, 1989.

[22] S. I. Gallant. The pocket algorithm for perceptron learning. Technical Report SG-85-20, Northeastern University College of Computer science, USA, 1985.

[23] J. Hampshire and B. Pearlmuter. Equivalence proofs for the multilayer perceptron classifier and the bayes discriminant function. In *Proceedings of the 1988 Connectionist Models Summer School*, San Mateo CA, 1990. Morgan Kaufmann.

[24] S. J. Hanson. Meiosis networks. In David S. Tourzetsky, editor, *Advances in Neural Information Processing Systems 2*, pages 533–541. Morgan Kaufmann, 1990.

[25] D. O. Hebb. *The Organisation of Behaviour.* John Wiley and Sons, New York, 1949.

[26] Markus Hoehfeld and Scott E. Fahlman. Learning with limited precision using the cascade-correlation algorithm. Technical Report CMU-CS-91-130, Carnegie Mellon University, USA, May 1991.

[27] R.A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307, 1988.

[28] T. Kohonen. *Self-Organization and Associative Memory.* Springer Verlag, Berlin, 1984.

[29] T. Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, Berlin, 3 edition, 1989.

[30] T. Kohonen, G. Barna, and R. Chrisley. Statistical pattern recognition with neural networks: Benchmarking studies. In *IEEE International Conference on Neural Networks*, volume 1, pages 61–68, New York, 1988. (San Diego 1988), IEEE.

[31] P.R. Krishnaiah and L.N. Kanal, editors. *Classification, Pattern Recognition, and Reduction of Dimensionality*, volume 2 of *Handbook of Statistics.* North Holland, Amsterdam, 1982.

[32] Kevin J. Lang, Alex H. Waibel, and Geoffrey E. Hinton. A time-delay neural network architecture for isolated word recognition. *Neural Networks*, 3:23–44, 1990.

[33] Y. Le Cun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

[34] Stephen P. Luttrell. Derivation of a class of training algorithms. *IEEE Transactions on Neural Networks*, 1(2):229–232, 1990.

[35] Stephen P. Luttrell. A bayesian analysis of vector quantization algorithms. *submitted to Neural Computation*, 1993.

[36] D. MacKay. The evidence framework applied to classification networks. *Neural Computation*, 4:720–736, 1992.

[37] D. MacKay. A practical bayesian framework for backpropagation networks. *Neural Computation*, 4:448–472, 1992.

[38] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity forms. *Bulletin of Methematical Biophysics*, 9:127–147, 1943.

[39] G. Meyer-Brötz and J. Schürmann. *Methoden der automatischen Zeichenerkennung.* Akademie-Verlag, Berlin, 1970.

[40] M. C. Minsky and S. Papert. *Perceptrons.* MIT Press, Cambridge, MA, USA, 1969.

[41] M. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 4:525–534, 1993.

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

[43] R. W. Prager and F. Fallside. The modified Kanerva model for automatic speech recognition. *Computer Speech and Language*, 3:61–82, 1989.

[44] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vettering. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, Cambridge, UK, 1988.

[45] A. N. Refenes and S. Vithlani. Constructive learning by specialisation. In *Proceedings of the International Conference on Artificial Neural Networks, Helsinki, Finland*, June 1991.

[46] Steve Renals and Richard Rohwer. Phoneme classification experiments using radial basis functions. In *Proceedings International Joint Conference on Neural Networks*, volume I, pages 461–468, Washington DC, 1989.

[47] R. Rohwer. Description and training of neural network dynamics. In F. Pasemann and H.D. Doebner, editors, *Neurodynamics, Proceedings of the 9th Summer Workshop, Clausthal, Germany.* World Scientific, 1991.

[48] R. Rohwer. Neural networks for time-varying data. In F. Murtagh, editor, *Neural Networks for Statistical and Economic Data*, pages 59–70, Luxembourg, 1991. Statistical Office of the European Communities.

[49] R. Rohwer. Time trials on second-order and variable-learning-rate algorithms. In R. Lippmann, J. Moody, and D. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 3, pages 977–983, San Mateo CA, 1991. Morgan Kaufmann.

[50] R. Rohwer and D. Cressy. Phoneme classification by boolean networks. In *Proceedings of the European Conference on Speech Communication and Technology*, pages 557–560, Paris, 1989.

[51] R. Rohwer, B. Grant, and P. R. Limb. Towards a connectionist reasoning system. *British Telecom Technology Journal*, 10:103–109, 1992.

[52] Richard Rohwer. A representation of representation applied to a discussion of variable binding. Technical report, Dept. of Computer Science and Applied Maths., Aston University, Birmingham B7 4ET, UK, 1992. To appear in Proc. Neurodynamics and Psychology Workshop, 22-24 April 1992, Bangor, Wales, M. Oaksford and G. Brown, Eds.

[53] Richard Rohwer and Steve Renals. Training recurrent networks. In L. Personnaz and G. Dreyfus, editors, *Neural networks from models to applications*, pages 207–216. I. D. S. E. T., Paris, 1988.

[54] F. Rosenblatt. *Psychological Review*, 65:368–408, 1958.

[55] F. Rosenblatt. *Principles of Neurodynamics.* Spartan Books, New York, 1958.

[56] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Procrressing*, volume 1, pages 318–362. MIT Press, Cambridge MA, 1986.

[57] R. Scalero and N. Tepedelenlioglu. A fast new algorithm for training feedforward neural networks. *IEEE Transactions on Signal Processing*, 40:202–210, 1992.

[58] L. Shastri and V. Ajjangadde. From simple associations to systematic reasoning: A connectionist representation of rules, variables, and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences*, to appear.

[59] Fernando M. Silva and Luis B. Almeida. Acceleration techniques for the backpropagation algorithm. In L. B. Almeida and C. J. Wellekens, editors, *Lecture Notes in Computer Science 412, Neural Networks*, pages 110–119. Springer-Verlag, Berlin, 1990.

[60] T. Toolenaere. Supersab: Fast adaptive back propagation with good scaling properties. *Neural Networks*, 3:561–574, 1990.

[61] S. Unger and F. Wysotzki. *Lernfähige Klassifizierungssysteme.* Akademie-Verlag, Berlin, 1981.

[62] P. Werbos. *Beyond Regression: New Tools for prediction and analysis in the behavioural sciences.* PhD thesis, Harvard University, 1975. Also printed as a report of the Harvard / MIT Cambridge Project, 1975.

[63] D. H. Wolpert. A rigorous investigation of "evidence" and "occam factors" in bayesian reasoning. Technical report, The Sante Fe Institute, 1660 Old Pecos Trail, Suite A, Sante Fe, NM, 87501, USA, 1992.

[64] J. X. Wu and C. Chan. A three layer adaptive network for pattern density estimation and classification. *International Journal of Neural Systems*, 2(3):211–220, 1991.

[65] Mike Wynne-Jones. Constructive algorithms and pruning: Improving the multi layer perceptron. In *Proceedings of IMACS '91, the 13th World Congress on Computation and Applied Mathematics, Dublin*, volume 2, pages 747–750, July 1991.

[66] Mike Wynne-Jones. Node splitting: A constructive algorithm for feed-forard neural networks. In John E. Moody, Steven J. Hanson, and Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 1072–1079. Morgan Kaufmann, 1992.

[67] Mike Wynne-Jones. Node splitting: A constructive algorithm for feed-forward neural networks. *Neural Computing and Applications*, 1(1):17–22, 1993.

[68] Jihon Yang and Vasant Honavar. Experiments with the cascade-correlation algorithm. Technical Report 91-16, Department of Computer Science, Iowa State University, 226 Atanasoff Hall, Ames, IA 50011-1040, USA, July 1991.