8

PRE-PROCESSING AND FEATURE EXTRACTION

Since neural networks can perform essentially arbitrary non-linear functional mappings between sets of variables, a single neural network could, in principle, be used to map the raw input data directly onto the required final output values. In practice, for all but the simplest problems, such an approach will generally give poor results for a number of reasons which we shall discuss below. For most applications it is necessary first to transform the data into some new representation before training a neural network. To some extent, the general-purpose nature of a neural network mapping means that less emphasis has to be placed on careful optimization of this pre-processing than would be the case with simple linear techniques, for instance. Nevertheless, in many practical applications the choice of pre-processing will be one of the most significant factors in determining the performance of the final system.

In the simplest case, pre-processing may take the form of a linear transformation of the input data, and possibly also of the output data (where it is sometimes termed post-processing). More complex pre-processing may involve reduction of the dimensionality of the input data. The fact that such dimensionality reduction can lead to improved performance may at first appear somewhat paradoxical, since it cannot increase the information content of the input data, and in most cases will reduce it. The resolution is related to the curse of dimensionality discussed in Section 1.4.

Another important way in which network performance can be improved, sometimes dramatically, is through the incorporation of *prior knowledge*, which refers to relevant information which might be used to develop a solution and which is additional to that provided by the training data. Prior knowledge can either be incorporated into the network structure itself or into the pre-processing and post-processing stages. It can also be used to modify the training process through the use of regularization, as discussed in Sections 9.2 and 10.1.2.

A final aspect of data preparation arises from the fact that real data often suffers from a number of deficiencies such as missing input values or incorrect target values.

In this chapter we shall focus primarily on classification problems. It should be emphasized, however, that most of the same general principles apply equally to regression problems.

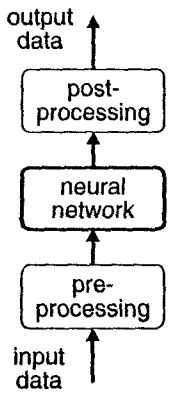

Figure 8.1. Schematic illustration of the use of data pre-processing and post-processing in conjunction with a neural network mapping.

8.1 Pre-processing and post-processing

In Chapter 1 we formulated the problem of pattern recognition in terms of a non-linear mapping from a set of input variables to a set of output variables. We have already seen that a feed-forward neural network can in principle represent an arbitrary functional mapping between spaces of many dimensions, and so it would appear that we could use a single network to map the raw input data directly onto the required output variables. In practice it is nearly always advantageous to apply pre-processing transformations to the input data before it is presented to a network. Similarly, the outputs of the network are often post-processed to give the required output values. These steps are indicated in Figure 8.1. The pre-processing and post-processing steps may consist of simple fixed transformations determined by hand, or they may themselves involve some adaptive processes which are driven by the data. For practical applications, data pre-processing is often one of the most important stages in the development of solution, and the choice of pre-processing steps can often have a significant effect on generalization performance.

Since the training of the neural network may involve an iterative algorithm, it will generally be convenient to process the whole training set using the pre-processing transformations, and then use this transformed data set to train the network. With applications involving on-line learning, each new data point must first be pre-processed before it is passed to the network. If post-processing of the network outputs is used, then the target data must be transformed using the inverse of the post-processing transformation in order to generate the target values for the network outputs. When subsequent data is processed by the trained network, it must first be passed through the pre-processing stage, then through the network, and finally through the post-processing transformation.

One of the most important forms of pre-processing involves a reduction in the dimensionality of the input data. At the simplest level this could involve discarding a subset of the original inputs. Other approaches involve forming linear or non-linear combinations of the original variables to generate inputs for the network. Such combinations of inputs are sometimes called *features*, and the process of generating them is called *feature extraction*. The principal motivation for dimensionality reduction is that it can help to alleviate the worst effects of the curse of dimensionality (Section 1.4). A network with fewer inputs has fewer adaptive parameters to be determined, and these are more likely to be properly constrained by a data set of limited size, leading to a network with better generalization properties. In addition, a network with fewer weights may be faster to train.

As a rather extreme example, consider the hypothetical character recognition problem discussed in Section 1.1. A 256×256 image has a total of 65 536 pixels. In the most direct approach we could take each pixel as the input to a single large neural network, which would give 65 537 adaptive weights (including the bias) for every unit in the first hidden layer. This implies that a very large training set would be needed to ensure that the weights were well determined, and this in turn implies that huge computational resources would be needed in order to find a suitable minimum of the error function. In practice such an approach is clearly impractical. One technique for dimensionality reduction in this case is *pixel averaging* which involves grouping blocks of pixels together and replacing each of them with a single effective pixel whose grey-scale value is given by the average of the grey-scale values of the original pixels in the block. It is clear that information is discarded by this process, and that if the blocks of pixels are too large, then there will be insufficient information remaining in the pixel averaged image for effective classification. These averaged pixels are examples of *features*, that is modified inputs formed from collections of the original inputs which might be combined in linear or non-linear ways. For an image interpretation problem it will often be possible to identify more appropriate features which retain more of the relevant information in the original image. For a medical classification problem, such features might include various measures of textures, while for a problem involving detecting objects in images, it might be more appropriate to extract features involving geometrical parameters such as the lengths of edges or the areas of contiguous regions.

Clearly in most situations a reduction in the dimensionality of the input vector will result in loss of information. One of the main goals in designing a good pre-processing strategy is to ensure that as much of the relevant information as possible is retained. If too much information is lost in the pre-processing stage then the resulting reduction in performance more than offsets any improvement arising from a reduction in dimensionality. Consider a classification problem in which an input vector $\mathbf{x}$ is to be assigned to one of c classes $\mathcal{C}_k$ where $k = 1, \ldots, c$. The minimum probability of misclassification is obtained by assigning each input vector $\mathbf{x}$ to the class $\mathcal{C}_k$ having the largest posterior probability $P(\mathcal{C}_k|\mathbf{x})$. We can regard these probabilities as examples of features. Since there are c such features,

and since they satisfy the relation $\sum_k P(\mathcal{C}_k|\mathbf{x}) = 1$, we see that in principle $c-1$ independent features are sufficient to give the optimal classifier. In practice, of course, we will not be able to obtain these probabilities easily, otherwise we would already have solved the problem. We may therefore need to retain a much larger number of features in order to ensure that we do not discard too much useful information. This discussion highlights the rather artificial distinction between the pre-processing stage and the classification or regression stage. If we can perform sufficiently clever pre-processing then the remaining operations become trivial. Clearly there is a balance to be found in the extent to which data processing is performed in the pre-processing and post-processing stages, and the extent to which it is performed by the network itself.

8.2 Input normalization and encoding

One of the most common forms of pre-processing consists of a simple linear rescaling of the input variables. This is often useful if different variables have typical values which differ significantly. In a system monitoring a chemical plant, for instance, two of the inputs might represent a temperature and a pressure respectively. Depending on the units in which each of these is expressed, they may have values which differ by several orders of magnitude. Furthermore, the typical sizes of the inputs may not reflect their relative importance in determining the required outputs.

By applying a linear transformation we can arrange for all of the inputs to have similar values. To do this, we treat each of the input variables independently, and for each variable x_i we calculate its mean $\overline{x}_i$ and variance σ_i^2 with respect to the training set, using

$$\overline{x}_i = \frac{1}{N}\sum_{n=1}^{N} x_i^n$$

$$\sigma_i^2 = \frac{1}{N-1}\sum_{n=1}^{N}(x_i^n - \overline{x}_i)^2 \tag{8.1}$$

where $n = 1, \ldots, N$ labels the patterns. We then define a set of re-scaled variables given by

$$\widetilde{x}_i^n = \frac{x_i^n - \overline{x}_i}{\sigma_i}. \tag{8.2}$$

It is easy to see that the transformed variables given by the $\widetilde{x}_i^n$ have zero mean and unit standard deviation over the transformed training set. In the case of regression problems it is often appropriate to apply a similar linear rescaling to the target values.

Note that the transformation in (8.2) is linear and so, for the case of a multilayer perceptron, it is in principle redundant since it could be combined with the linear transformation in the first layer of the network. In practice, however, input normalization ensures that all of the input and target variables are of order unity, in which case we expect that the network weights should also be of order unity. The weights can then be given a suitable random initialization prior to network training. Without the linear rescaling, we would need to find a solution for the weights in which some weight values had markedly different values from others.

Note that, in the case of a radial basis function network with spherically-symmetric basis functions, it is particularly important to normalize the input variables so that they span similar ranges. This is a consequence of the fact that the activation of a basis function is determined by the Euclidean distance l between the input vector $\mathbf{x}$ and the basis function centre $\boldsymbol{\mu}_j$ given by

$$l^2 = \|\mathbf{x} - \boldsymbol{\mu}_j\|^2 = \sum_{i=1}^{d} \{x_i - \mu_{ji}\}^2 \tag{8.3}$$

where d is the dimensionality of the input space. If one of the input variables has a much smaller range of values than the others, the value of l^2 will be very insensitive to this variable. In principle, an alternative to normalization of the input data is to use basis functions with more general covariance matrices.

The simple linear rescaling in (8.2) treats the variables as independent. We can perform a more sophisticated linear rescaling, known as *whitening*, which allows also for correlations amongst the variables (Fukunaga, 1990). For convenience we group the input variables x_i into a vector $\mathbf{x} = (x_1, \ldots, x_d)^{\mathrm{T}}$, which has sample mean vector and covariance matrix with respect to the N data points of the training set given by

$$\overline{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}^n$$

$$\boldsymbol{\Sigma} = \frac{1}{N-1} \sum_{n=1}^{N} (\mathbf{x}^n - \overline{\mathbf{x}})(\mathbf{x}^n - \overline{\mathbf{x}})^{\mathrm{T}}. \tag{8.4}$$

If we introduce the eigenvalue equation for the covariance matrix

$$\boldsymbol{\Sigma}\mathbf{u}_j = \lambda_j \mathbf{u}_j \tag{8.5}$$

then we can define a vector of linearly transformed input variables given by

$$\widetilde{\mathbf{x}}^n = \boldsymbol{\Lambda}^{-1/2}\mathbf{U}^{\mathrm{T}}(\mathbf{x}^n - \overline{\mathbf{x}}) \tag{8.6}$$

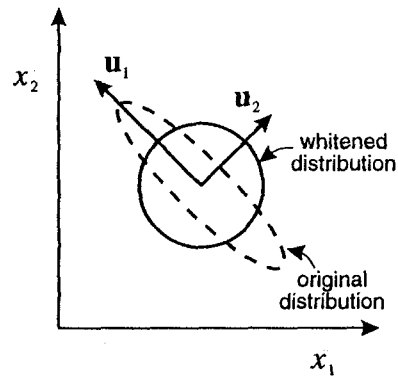

Figure 8.2. Schematic illustration of the use of the eigenvectors $\mathbf{u}_j$ (together with their corresponding eigenvalues λ_j) of the covariance matrix of a distribution to whiten the distribution so that its covariance matrix becomes the unit matrix.

where we have defined

$$\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_d) \tag{8.7}$$

$$\mathbf{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_d). \tag{8.8}$$

Then it is easy to verify that, in the transformed coordinates, the data set has zero mean and a covariance matrix which is given by the unit matrix. This is illustrated schematically in Figure 8.2.

8.2.1 *Discrete data*

So far we have discussed data which takes the form of continuous variables. We may also have to deal with data taking on discrete values. In such cases it is convenient to distinguish between *ordinal* variables which have a natural ordering, and *categorical* variables which do not. An example of an ordinal variable would be a person's age in years. Such data can simply be transformed directly into the corresponding values of a continuous variable. An example of a categorical variable would be a measurement which could take one of the values red, green or blue. If these were to be represented as, for instance, the values 0.0, 0.5 and 1.0 of a single continuous input variable, this would impose an artificial ordering on the data. One way around this is to use a 1-of-c coding for the input data, similar to that discussed for target data in classification problems in Section 6.6. In the above example this requires three input variables, with the three colours represented by input values of $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$.

8.3 Missing data

In practical applications it sometimes happens that the data suffers from deficiencies which should be remedied before the data is used for network training. A common problem is that some of the input values may be missing from the data set for some of the pattern vectors (Little and Rubin, 1987; Little, 1992). If the quantity of data available is sufficiently large, and the proportion of patterns affected is small, then the simplest solution is to discard those patterns from the data set. Note that this approach is implicitly assuming that the mechanism which is responsible for the omission of data values is independent of the data itself. If the values which are missing depend on the data, then this approach will modify the effective data distribution. An example would be a sensor which always fails to produce an output signal when the signal value exceeds some threshold.

When there is too little data to discard the deficient examples, or when the proportion of deficient points is too high, it becomes important to make full use of the information which is potentially available from the incomplete patterns. Consider first the problem of unconditional density estimation, for the case of a parametric model based on a single Gaussian distribution. A common heuristic for estimating the model parameters would be the following. The components μ_i of the mean vector $\boldsymbol{\mu}$ are estimated from the values of x_i for all of the data points for which this value is available, irrespective of whether other input values are present. Similarly, the (i, j) element of the covariance matrix $\boldsymbol{\Sigma}$ is found using all pairs of data points for which values of both x_i and x_j are available. Such an approach, however, can lead to poor results (Ghahramani and Jordan, 1994b), as indicated in Figure 8.3.

Various heuristics have also been proposed for dealing with missing input data in regression and classification problems. For example, it is common to 'fill in' the missing input values first (Hand, 1981), and then train a feed-forward network using some standard method. For example, each missing value might be replaced by the mean of the corresponding variable over those patterns for which its value is available. This is prone to serious problems as discussed above. A more elaborate approach is to express any variable which has missing values in terms of a regression over the other variables using the available data, and then to use the regression function to fill in the missing values. Again, this approach tends to cause problems as it underestimates the covariance in the data since the regression function is noise-free.

Missing data in density estimation problems can be dealt with in a principled way by seeking a maximum likelihood solution, and using the expectation–maximization, or EM, algorithm to deal with missing data. In Section 2.6.2, the EM algorithm was introduced as a technique for finding maximum likelihood solutions for mixture models, in which hypothetical variables describing which component was responsible for generating each data point were introduced and treated as 'missing data'. The EM algorithm can similarly be applied to the problem of variables missing from the data itself (Ghahramani and Jordan, 1994b).

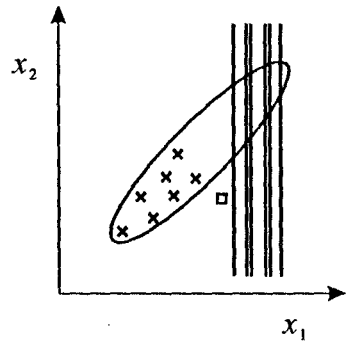

Figure 8.3. Schematic illustration of a set of data points in two dimensions. For some of the data points (shown by the crosses) the values of both variables are present, while for others (shown by the vertical lines) only the values of x_1 are known. If the mean vector of the distribution is estimated using the available values of each variable separately, then the result is a poor estimate, as indicated by the square.

In fact the two problems can be tackled together, so that the parameters of a mixture model can be estimated, even when there is missing data. Such techniques can be applied to the determination of the basis function parameters in a radial basis function network, as discussed in Section 5.9.4. They can also be used to determine the density $p(\mathbf{x}, \mathbf{t})$ in the joint input-target space. From this density, the conditional density $p(\mathbf{t}|\mathbf{x})$ can be evaluated, as can the regression function $\langle \mathbf{t}|\mathbf{x} \rangle$.

In general, missing values should be treated by integration over the corresponding variables (Ahmad and Tresp, 1993), weighted by the appropriate distribution (Exercise 8.4). This requires that the input distribution itself be modelled. A related approach is to fill in the missing data points with values drawn at random from this distribution (Lowe and Webb, 1990). It is then possible to generate many different 'completions' of a given input pattern which has missing variables. This can be regarded as a simple Monte Carlo approximation to the required integration over the input distribution (Section 10.9).

8.4 Time series prediction

Many potential applications of neural networks involve data $\mathbf{x} = \mathbf{x}(\tau)$ which varies as a function of time τ. The goal is often to predict the value of $\mathbf{x}$ a short time into the future. Techniques based on feed-forward networks, of the kind described in earlier chapters, can be applied directly to such problems provided the data is appropriately pre-processed first. Consider for simplicity a single variable $x(\tau)$. One common approach is to sample $x(\tau)$ at regular intervals to generate a series of discrete values $x_{\tau-1}, x_\tau, x_{\tau+1}$ and so on. We can take a set

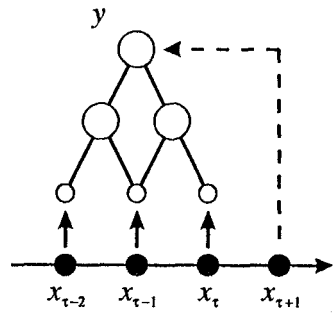

Figure 8.4. Sampling of a time series at discrete steps can be used to generate a set of training data for a feed-forward network. Successive values of the time-dependent variable $x(\tau)$, given by $x_{\tau-d+1}, \ldots, x_{\tau}$, form the inputs to a feed-forward network, and the corresponding target value is given by $x_{\tau+1}$.

of d such values $x_{\tau-d+1}, \ldots, x_{\tau}$ to be the inputs to a feed-forward network, and use the next value $x_{\tau+1}$ as the target for the output of the network, as indicated in Figure 8.4. By stepping along the time axis, we can create a training data set consisting of many sets of input values with corresponding target values. Once the network has been trained, it can be presented with a set of observed values $x_{\tau'-d+1}, \ldots, x_{\tau'}$ and used to make a prediction for $x_{\tau'+1}$. This is called *one step ahead* prediction. If the predictions themselves are cycled around to the inputs of the network, then predictions can be made at further points $x_{\tau'+2}$ and so on. This is called *multi-step ahead* prediction, and is typically characterized by a rapidly increasing divergence between the predicted and observed values as the number of steps ahead is increased due to the accumulation of errors. The above approach is easily generalized to deal with several time-dependent variables in the form of a time-dependent vector $\mathbf{x}(\tau)$.

One drawback with this technique is the need to choose the time increment between successive inputs, and this may require some empirical optimization. Another problem is that the time series may show an underlying trend, such as a steadily increasing value, with more complex structure superimposed. This can be removed by fitting a simple (e.g. linear) function of time to the data, and then subtracting off the predictions of this simple model. Such pre-processing is called *de-trending*, and without it, a trained network would be forced to extrapolate when presented with new data, and would therefore have poor performance.

There is a key assumption which is implicit in this approach to time series prediction, which is that the statistical properties of the generator of the data (after de-trending) are time-independent. Provided this is the case, then the pre-processing described above has mapped the time series problem onto a static function approximation problem, to which a feed-forward network can be applied.

If, however, the generator of the data itself evolves with time, then this approach is inappropriate and it becomes necessary for the network model to adapt to the data continuously so that it can 'track' the time variation. This requires on-line learning techniques, and raises a number of important issues, many of which are at present largely unresolved and lie outside the scope of this book.

8.5 Feature selection

One of the simplest techniques for dimensionality reduction is to select a subset of the inputs, and to discard the remainder. This approach can be useful if there are inputs which carry little useful information for the solution of the problem, or if there are very strong correlations between sets of inputs so that the same information is repeated in several variables. It can be applied not only to the original data, but also to a set of candidate features constructed by some other means. For convenience we shall talk of feature selection, even though the features might simply be the original input variables. Many of the ideas are equally applicable to conventional approaches to pattern recognition, and are covered in a number of the standard books in this area including Hand (1981), Devijver and Kittler (1982) and Fukunaga (1990), and are reviewed in Siedlecki and Sklansky (1988).

Any procedure for feature selection must be based on two components. First, a criterion must be defined by which it is possible to judge whether one subset of features is better than another. Second, a systematic procedure must be found for searching through candidate subsets of features. In principle the selection criterion should be the same as will be used to assess the complete system (such as misclassification rate for a classification problem or sum-of-squares error for a regression problem). Similarly, the search procedure could simply consist of an exhaustive search of all possible subsets of features since this is in general the only approach which is guaranteed to find the optimal subset. In a practical application, however, we are often forced to consider simplified selection criteria as well as non-exhaustive search procedures in order to limit the computational complexity of the search process. We begin with a discussion of possible selection criteria.

8.5.1 *Selection criteria*

It is clear that the optimal subset of features selected from a given starting set will depend, among other things, on the particular form of model (neural network or otherwise) with which they are to be used. Ideally the selection criterion would be obtained by training the network on the given subset of features, and then evaluating its performance on an independent set of test data. If the network training procedure involves non-linear optimization, such an approach is likely to be impractical since the training and testing process would have to be repeated for each new choice of feature subset, and the computational requirements would become too great. It is therefore common to use a simpler model, such as a linear mapping, in order to select the features, and then use these features with the more sophisticated non-linear model. The simplified model is chosen so that it can

be trained relatively quickly (using linear matrix methods for instance) thereby permitting a relatively large number of feature combinations to be explored. It should be emphasized, however, that the feature selection and the classification (or regression) stages should be ideally be optimized together, and that it is only because of practical constraints that we are often forced to treat them independently.

For regression problems, we can take the simple model to be a linear mapping given by a single-layer network with linear output units, which is equivalent to matrix multiplication with the addition of a bias vector. If the error function for network training is given by a sum-of-squares, we can use this same measure for feature selection. In this case, the optimal values for the weights and biases in the linear mapping can be expressed in terms of a set of linear equations whose solution can be found quickly by using singular value decomposition (Section 3.4.3).

For classification problems, the selection criterion should ideally be taken to be the probability of misclassification, or more generally as the expected total loss or risk. This could in principle be calculated by using either parametric or non-parametric techniques to estimate the posterior probabilities for each class (Hand, 1981). In practice, evaluation of this criterion directly is generally too complex, and we have to resort instead to simpler criteria such as those based on class separability. We expect that a set of variables in which the classes are best separated will be a good set of variables for input to a neural network or other classifier. Appropriate criteria for class separability, based on covariance matrices, were discussed in Section 3.6 in the context of the Fisher discriminant and its generalizations.

If we were able to use the full criterion of misclassification rate, we would expect that, as we reduce the number of features which are retained, the generalization performance of the system would improve (a consequence of the curse of dimensionality) until some optimal subset of features is reached, and that if fewer features are retained the performance will degrade. One of the limitations of many simple selection criteria, such as those based on class separability, is that they are incapable of modelling this phenomenon. For example, the Mahalanobis distance Δ^2 (Section 2.1.1) always increases as extra variables are added. In general such measures J satisfy a monotonicity property such that

$$J(X^+) \geq J(X) \tag{8.9}$$

where X denotes a set of features, and X^+ denotes a larger set of features which contains the set X as a subset. This property is shared by criteria based on covariance matrices. The inequality simply says that deleting features cannot reduce the error rate. As a consequence, criteria which satisfy the monotonicity constraint cannot be used to determine the optimum size for a set of variables and so cannot be used to compare sets of different sizes. However, they do offer a useful way to compare sets of variables having the same number of elements. In

practice the removal of features can improve the error rate when we take account of the effects of a finite size data set. One approach to the set size problem is to use conventional statistical tests to measure the significance of the improvement in discrimination resulting from inclusion of extra variables (Hand, 1981). Another approach is to apply cross-validation techniques (Section 9.8.1) to compare models trained using different numbers of features, where the particular feature subset used for each model is determined by one of the approaches discussed here.

8.5.2 *Search procedures*

If we have a total of d possible features, then since each feature can be present or absent, there are a total of 2^d possible feature subsets which could be considered. For a relatively small number of features we might consider simply using exhaustive search. With 10 input variables, for example, there are 1024 possible subsets which it might be computationally feasible to consider. For large numbers of input variables, however, exhaustive search becomes prohibitively expensive. Thus with 100 inputs there are over 10^{30} possible subsets, and exhaustive search is impossible. If we have already decided that we want to extract precisely $\tilde{d}$ features then the number of combinations of features is given by

$$\frac{d!}{(d-\tilde{d})!\tilde{d}!} \tag{8.10}$$

which can be significantly smaller than 2^d, but which may still be impractically large in many applications.

In principle it may be necessary to consider all possible subsets of features, since combinations of variables can provide significant information which is not available in any of the individual variables separately. This is illustrated for two classes, and two features x_1 and x_2, in Figure 8.5. Either feature taken alone gives strong overlap between the two classes, while if the two features are considered together then the classes form well-separated clusters. A similar effect can occur with an arbitrary number of features so that, in the most general case, the only way to find the optimum subset is to perform exhaustive search.

If we are using a criterion which satisfies the monotonicity relation in (8.9) then there exists an accelerated search procedure known as *branch and bound* (Narendra and Fukunaga, 1977). This method can also be applied in many other areas such as cluster analysis and searching for nearest neighbours. In the present context it will guarantee to find the best subset of given size, without needing to evaluate all possible subsets. To understand this technique, we begin by discussing the exhaustive search procedure, which we set out as a tree structure. Consider an original set of d features x_i where $i = 1, \ldots, d$, and denote the indices of the $M = d - \tilde{d}$ features which have been discarded by $z_1, \ldots, z_M$, where each z_k can take the value $1, \ldots, d$. However, no two z_k should take the same value, since that would represent a single feature being eliminated twice.

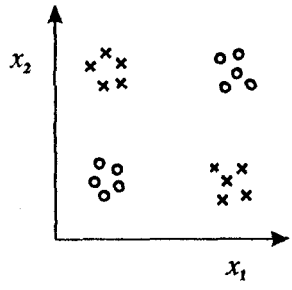

Figure 8.5. Example of data from two classes (represented by the crosses and the circles respectively) as described by two feature variables x_1 and x_2. If the data was described by either feature alone then there would be strong overlap of the two classes, while with if both features are used, as shown here, then the classes are well separated.

Also, the order of the z_k's is irrelevant in defining the feature subset. A sufficient condition for satisfying these constraints is that the z_k should satisfy

$$z_1 < z_2 < \ldots < z_M. \tag{8.11}$$

This allows us to construct a search tree, as shown in Figure 8.6 for the case of five original features from which we wish to select a subset of two. The features are indexed by the labels 1, 2, 3, 4, 5, and the number next to each node denotes the feature which is eliminated at that node. Each possible subset of two features selected from a total of five is represented by one of the nodes at the bottom of the tree. At the first level down from the top of the tree, the highest value of z_k which is considered is 3, since any higher value would not allow the constraint (8.11) to be satisfied. Similar arguments are used to construct the rest of the tree. Now suppose that we wish to maximize a criterion $J(\widetilde{d})$ and that the value of J corresponding to the node shown at A is recorded as a threshold. If at any point in the search an intermediate node is encountered, such as that shown at B, for which the value of J is smaller than the threshold, then there is no need to evaluate any of the sets which lie below this node on the tree, since, as a consequence of the monotonicity relation (8.9), such nodes necessarily have values of the criterion which are smaller than the threshold. Thus, the nodes shown as solid circles in Figure 8.6 need not be evaluated. If at any point in the search a final-layer node is encountered which has a larger value for the criterion, then this value becomes the new threshold. The algorithm terminates when every final-layer node has either been evaluated or excluded using the monotonicity relation. Note that, unlike exhaustive search applied to all possible subsets of $\widetilde{d}$ variables, this method requires evaluation of some of the intermediate sub-sets

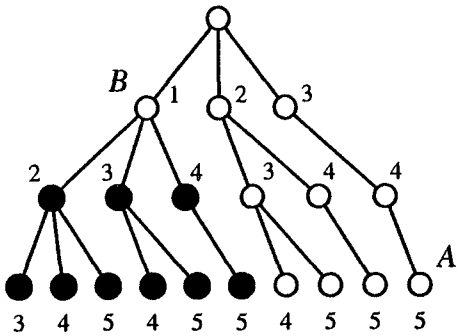

Figure 8.6. A search tree for feature subset selection, for the case of a set of five feature variables from which we wish to pick out the optimum subset of two variables. If a strictly monotonic selection criterion is being used, and a node such as that at B is found which has a lower value for the criterion than some final-level node such as that at A, then all nodes below B (shown as solid black nodes) can be eliminated from the search.

which contain more than $\widetilde{d}$ variables. However, this is more than offset by the savings in not having to evaluate final-layer subsets which are excluded using the monotonicity property. The basic branch and bound algorithm can be modified to generate a tree in which nodes with smaller values of the selection criterion tend to have larger numbers of successive branches (Fukunaga, 1990). This can lead to improvements in computational efficiency since nodes with smaller values of the criterion are more likely to be eliminated from the search tree.

8.5.3 *Sequential search techniques*

The branch and bound algorithm for monotonic selection criteria is generally faster than exhaustive search but is still guaranteed to find the feature subset (of given size) which maximizes the criterion. In some applications, such an approach is still computationally too expensive, and we are then forced to consider techniques which are significantly faster but which may give suboptimal solutions. The simplest method would be to select those $\widetilde{d}$ features which are individually the best (obtained by evaluating the selection criterion using one feature at a time). This method, however, is likely to be highly unreliable, and would only be optimal for selection criteria which can be expressed as the sum, or the product, of the criterion evaluated for each feature individually, and it would therefore only be appropriate if the features were completely independent.

A better approach, known as *sequential forward selection*, is illustrated in Figure 8.7. The procedure begins by considering each of the variables individually and selecting the one which gives the largest value for the selection criterion. At each successive stage of the algorithm, one additional feature is added to the set,

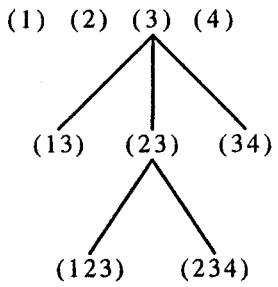

Figure 8.7. Sequential forward selection illustrated for a set of four input features, denoted by 1, 2, 3 and 4. The single best feature variable is chosen first, and then features are added one at a time such that at each stage the variable chosen is the one which produces the greatest increase in the criterion function.

again chosen on the basis of which of the possible candidates at that stage gives rise to the largest increase in the value of the selection criterion. One obvious difficulty with this approach is that, if there are two feature variables of the kind shown in Figure 8.5, such that either feature alone provides little discrimination, but where both features together are very effective, then the forward selection procedure may never find this combination since either feature alone would never be selected.

An alternative is to start with the full set of d features and to eliminate them one at a time. This gives rise to the technique of *sequential backward elimination* illustrated in Figure 8.8. At each stage of the algorithm, one feature is deleted from the set, chosen from amongst all available candidates as the one which gives the smallest reduction in the value of the selection criterion. This overcomes the problem with the forward selection approach highlighted above, but is still not guaranteed to be optimal. The backward elimination algorithm requires a greater number of evaluations, however, since it considers numbers of features greater than or equal to $\widetilde{d}$ while the forward selection procedure considers numbers of features less than or equal to $\widetilde{d}$.

These algorithms can be generalized in various ways in order to allow small subsets of features which are collectively useful to be selected (Devijver and Kittler, 1982). For example, at the kth stage of the algorithm, we can add l features using the sequential forward algorithm and then eliminate r features using the sequential backwards algorithm. Clearly there are many variations on this theme giving a range of algorithms which search a larger range of feature subsets at the price of increased computation.

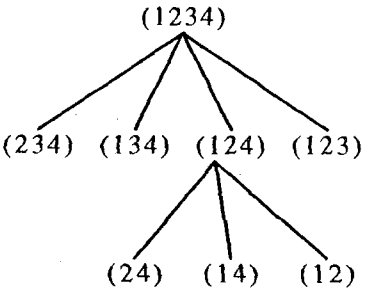

Figure 8.8. Sequential backward elimination of variables, again illustrated for the case of four features. Starting with the complete set, features are eliminated one at a time, such that at each stage the feature chosen for elimination is the one corresponding to the smallest reduction in the value of the selection criterion.

8.6 Principal component analysis

We have already discussed the problems which can arise in attempts to perform pattern recognition in high-dimensional spaces, and the potential improvements which can be achieved by first mapping the data into a space of lower dimensionality. In general, a reduction in the dimensionality of the input space will be accompanied by a loss of some of the information which discriminates between different classes (or, more generally, which determines the target values). The goal in dimensionality reduction is therefore to preserve as much of the relevant information as possible. We have already discussed one approach to dimensionality reduction based on the selection of a subset of a given set of features or inputs. Here we consider techniques for combining inputs together to make a (generally smaller) set of features. The procedures we shall discuss in this section rely entirely on the input data itself without reference to the corresponding target data, and can be regarded as a form of *unsupervised* learning. While they are of great practical significance, the neglect of the target data information implies they can also be significantly sub-optimal, as we discuss in Section 8.6.3.

We begin our discussion of unsupervised techniques for dimensionality reduction by restricting our attention to linear transformations. Our goal is to map vectors $\mathbf{x}^n$ in a d-dimensional space $(x_1, \ldots, x_d)$ onto vectors $\mathbf{z}^n$ in an M-dimensional space $(z_1, \ldots, z_M)$, where $M < d$. We first note that the vector $\mathbf{x}$ can be represented, without loss of generality, as a linear combination of a set of d orthonormal vectors $\mathbf{u}_i$

$$\mathbf{x} = \sum_{i=1}^{d} z_i \mathbf{u}_i \tag{8.12}$$

where the vectors $\mathbf{u}_i$ satisfy the orthonormality relation

$$\mathbf{u}_i^{\mathrm{T}}\mathbf{u}_j = \delta_{ij} \tag{8.13}$$

in which δ_{ij} is the Kronecker delta symbol defined on page xiii. Explicit expressions for the coefficients z_i in (8.12) can be found by using (8.13) to give

$$z_i = \mathbf{u}_i^{\mathrm{T}}\mathbf{x} \tag{8.14}$$

which can be regarded as a simple rotation of the coordinate system from the original x's to a new set of coordinates given by the z's (Appendix A). Now suppose that we retain only a subset $M < d$ of the basis vectors $\mathbf{u}_i$, so that we use only M coefficients z_i. The remaining coefficients will be replaced by constants b_i so that each vector $\mathbf{x}$ is approximated by an expression of the form

$$\widetilde{\mathbf{x}} = \sum_{i=1}^{M} z_i\mathbf{u}_i + \sum_{i=M+1}^{d} b_i\mathbf{u}_i. \tag{8.15}$$

This represents a form of dimensionality reduction since the original vector $\mathbf{x}$ which contained d degrees of freedom must now be approximated by a new vector $\mathbf{z}$ which has $M < d$ degrees of freedom. Now consider a whole data set of N vectors $\mathbf{x}^n$ where $n = 1, \ldots, N$. We wish to choose the basis vectors $\mathbf{u}_i$ and the coefficients b_i such that the approximation given by (8.15), with the values of z_i determined by (8.14), gives the best approximation to the original vector $\mathbf{x}$ on average for the whole data set. The error in the vector $\mathbf{x}^n$ introduced by the dimensionality reduction is given by

$$\mathbf{x}^n - \widetilde{\mathbf{x}}^n = \sum_{i=M+1}^{d} (z_i^n - b_i)\mathbf{u}_i. \tag{8.16}$$

We can then define the best approximation to be that which minimizes the sum of the squares of the errors over the whole data set. Thus, we minimize

$$E_M = \frac{1}{2}\sum_{n=1}^{N} \|\mathbf{x}^n - \widetilde{\mathbf{x}}^n\|^2 = \frac{1}{2}\sum_{n=1}^{N}\sum_{i=M+1}^{d} (z_i^n - b_i)^2 \tag{8.17}$$

where we have used the orthonormality relation (8.13). If we set the derivative of E_M with respect to b_i to zero we find

$$b_i = \frac{1}{N}\sum_{n=1}^{N} z_i^n = \mathbf{u}_i^{\mathrm{T}}\overline{\mathbf{x}} \tag{8.18}$$

where we have defined the mean vector $\overline{\mathbf{x}}$ to be

$$\overline{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}^n. \tag{8.19}$$

Using (8.14) and (8.18) we can write the sum-of-squares error (8.17) as

$$E_M = \frac{1}{2}\sum_{i=M+1}^{d}\sum_{n=1}^{N}\left\{\mathbf{u}_i^{\mathrm{T}}(\mathbf{x}^n - \overline{\mathbf{x}})\right\}^2$$

$$= \frac{1}{2}\sum_{i=M+1}^{d}\mathbf{u}_i^{\mathrm{T}}\boldsymbol{\Sigma}\mathbf{u}_i \tag{8.20}$$

where $\boldsymbol{\Sigma}$ is the covariance matrix of the set of vectors $\{\mathbf{x}^n\}$ and is given by

$$\boldsymbol{\Sigma} = \sum_n (\mathbf{x}^n - \overline{\mathbf{x}})(\mathbf{x}^n - \overline{\mathbf{x}})^{\mathrm{T}}. \tag{8.21}$$

There now remains the task of minimizing E_M with respect to the choice of basis vectors $\mathbf{u}_i$. It is shown in Appendix E that the minimum occurs when the basis vectors satisfy

$$\boldsymbol{\Sigma}\mathbf{u}_i = \lambda_i\mathbf{u}_i \tag{8.22}$$

so that they are the eigenvectors of the covariance matrix. Note that, since the covariance matrix is real and symmetric, its eigenvectors can indeed be chosen to be orthonormal as assumed. Substituting (8.22) into (8.20), and making use of the orthonormality relation (8.13), we obtain the value of the error criterion at the minimum in the form

$$E_M = \frac{1}{2}\sum_{i=M+1}^{d}\lambda_i. \tag{8.23}$$

Thus, the minimum error is obtained by choosing the $d-M$ smallest eigenvalues, and their corresponding eigenvectors, as the ones to discard.

The linear dimensionality reduction procedure derived above is called the *Karhunen–Loéve transformation* or *principal component analysis* and is discussed at length in Jollife (1986). Each of the eigenvectors $\mathbf{u}_i$ is called a *principal component.* The technique is illustrated schematically in Figure 8.9 for the case of data points in two dimensions.

In practice, the algorithm proceeds by first computing the mean of the vectors $\mathbf{x}^n$ and then subtracting off this mean. Then the covariance matrix is calculated

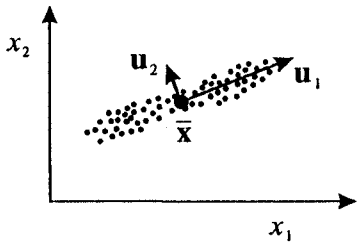

Figure 8.9. Schematic illustration of principal component analysis applied to data in two dimensions. In a linear projection down to one dimension, the optimum choice of projection, in the sense of minimizing the sum-of-squares error, is obtained by first subtracting off the mean $\bar{\mathbf{x}}$ of the data set, and then projecting onto the first eigenvector $\mathbf{u}_1$ of the covariance matrix.

and its eigenvectors and eigenvalues are found. The eigenvectors corresponding to the M *largest* eigenvalues are retained and the input vectors $\mathbf{x}^n$ are projected onto the eigenvectors to give the components of the transformed vectors $\mathbf{z}^n$ in the M-dimensional space. Thus, in Figure 8.9, each two-dimensional data point is transformed to a single variable z_1 representing the projection of the data point onto the eigenvector $\mathbf{u}_1$.

The error introduced by a dimensionality reduction using principal component analysis can be evaluated using (8.23). In some applications the original data has a very high dimensionality and we wish only to retain the first few principal components. In such cases use can be made of efficient algorithms which allow only the required eigenvectors, corresponding to the largest few eigenvalues, to be evaluated (Press *et al.*, 1992).

We have considered linear dimensionality reduction based on the sum-of-squares error criterion. It is possible to consider other criteria including data covariance measures and population entropy. These give rise to the same result for the optimal dimensionality reduction in terms of projections onto the eigenvectors of $\boldsymbol{\Sigma}$ corresponding to the largest eigenvalues (Fukunaga, 1990).

8.6.1 *Intrinsic dimensionality*

Suppose we are given a set of data vectors in a d-dimensional space, and we apply principal component analysis and discover that the first d' eigenvalues have significantly larger values than the remaining $d-d'$ eigenvalues. This tells us that the data can be represented to a relatively high accuracy by projection onto the first d' eigenvectors. We therefore discover that the effective dimensionality of the data is less than the apparent dimensionality d, as a result of correlations within the data. However, principal component analysis is limited by virtue of being a linear technique. It may therefore be unable to capture more complex non-linear correlations, and may therefore overestimate the true dimensionality

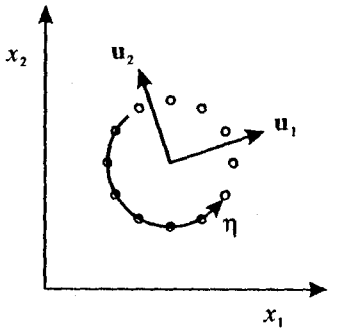

Figure 8.10. Example of a data set in two dimensions which has an intrinsic dimensionality $d' = 1$. The data can be specified not only in terms of the two variables x_1 and x_2, but also in terms of the single parameter η. However, a linear dimensionality reduction technique, such as principal component analysis, is unable to detect the lower dimensionality.

of the data. This is illustrated schematically in Figure 8.10, for data points which lie around the perimeter of a circle. Principal component analysis would give two eigenvectors with equal eigenvalues (as a result of the symmetry of the data). In fact, however, the data could be described equally well by a single parameter η as shown. More generally, a data set in d dimensions is said to have an *intrinsic dimensionality* equal to d' if the data lies entirely within a d'-dimensional subspace (Fukunaga, 1982).

Note that if the data is slightly noisy, then the intrinsic dimensionality may be increased. Figure 8.11 shows some data in two dimensions which is corrupted by a small level of noise. Strictly the data now lives in a two-dimensional space, but can nevertheless by represented to high accuracy by a single parameter.

8.6.2 *Neural networks for dimensionality reduction*

Multi-layer neural networks can themselves be used to perform non-linear dimensionality reduction, thereby overcoming some of the limitations of linear principal component analysis. Consider first a multi-layer perceptron of the form shown in Figure 8.12, having d inputs, d output units and M hidden units, with $M < d$ (Rumelhart *et al.*, 1986). The targets used to train the network are simply the input vectors themselves, so that the network is attempting to map each input vector onto itself. Due to the reduced number of units in the first layer, a perfect reconstruction of all input vectors is not in general possible. The network can be trained by minimizing a sum-of-squares error of the form

$$E = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{d} \{y_k(\mathbf{x}^n) - x_k^n\}^2. \tag{8.24}$$

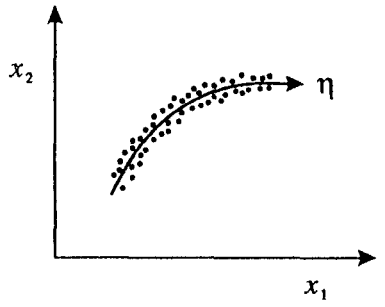

Figure 8.11. Addition of a small level of noise to data in two dimensions having an intrinsic dimensionality of 1 can increase its intrinsic dimensionality to 2. Nevertheless, the data can be represented to a good approximation by a single variable η and for practical purposes can be regarded as having an intrinsic dimensionality of 1.

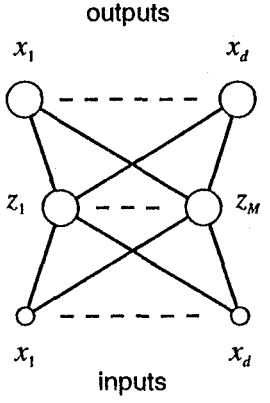

Figure 8.12. An auto-associative multi-layer perceptron having two layers of weights. Such a network is trained to map input vectors onto themselves by minimization of a sum-of-squares error. Even with non-linear units in the hidden layer, such a network is equivalent to linear principal component analysis. Biases have been omitted for clarity.

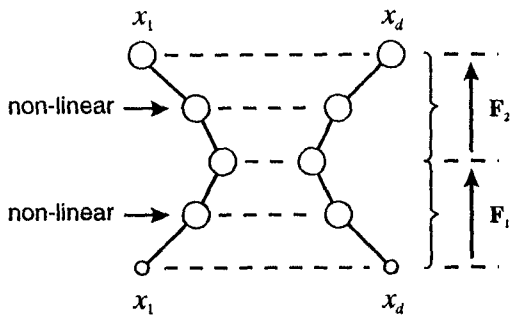

Figure 8.13. Addition of extra hidden layers of non-linear units to the network of Figure 8.12 gives an auto-associative network which can perform a general non-linear dimensionality reduction. Biases have been omitted for clarity.

Such a network is said to form an *auto-associative* mapping. Error minimization in this case represents a form of unsupervised training, since no independent target data is provided. If the hidden units have linear activations functions, then it can be shown that the error function has a unique global minimum, and that at this minimum the network performs a projection onto the M-dimensional sub-space which is spanned by the first M principal components of the data (Bourlard and Kamp, 1988; Baldi and Hornik, 1989). Thus, the vectors of weights which lead into the hidden units in Figure 8.12 form a basis set which spans the principal sub-space. (Note, however, that these vectors need not be orthogonal or normalized.) This result is not surprising, since both principal component analysis and the neural network are using linear dimensionality reduction and are minimizing the same sum-of-squares error function.

It might be thought that the limitations of a linear dimensionality reduction could be overcome by using non-linear (sigmoidal) activation functions for the hidden units in the network in Figure 8.12. However, it was shown by Bourlard and Kamp (1988) that such non-linearities make no difference, and that the minimum error solution is again given by the projection onto the principal component sub-space. There is therefore no advantage in using two-layer neural networks to perform dimensionality reduction. Standard techniques for principal component analysis (based on singular value decomposition) are guaranteed to give the correct solution in finite time, and also generate an ordered set of eigenvalues with corresponding orthonormal eigenvectors.

The situation is different, however, if additional hidden layers are permitted in the network. Consider the four-layer auto-associative network shown in Figure 8.13. Again the output units are linear, and the M units in the second hidden layer can also be linear. However, the first and third hidden layers have sigmoidal non-linear activation functions. The network is again trained by min-

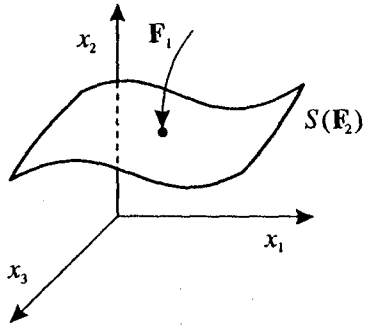

Figure 8.14. Geometrical interpretation of the mappings performed by the network in Figure 8.13.

imization of the error in (8.24). We can view this network as two successive functional mappings $\mathbf{F}_1$ and $\mathbf{F}_2$. The first mapping $\mathbf{F}_1$ projects the original d-dimensional data onto an M-dimensional sub-space $\mathcal{S}$ defined by the activations of the units in the second hidden layer. Because of the presence of the first hidden layer of non-linear units, this mapping is essentially arbitrary, and in particular is not restricted to being linear. Similarly the second half of the network defines an arbitrary functional mapping from the M-dimensional space back into the original d-dimensional space. This has a simple geometrical interpretation, as indicated for the case $d = 3$ and $M = 2$ in Figure 8.14. The function $\mathbf{F}_2$ maps from an M-dimensional space $\mathcal{S}$ into a d-dimensional space and therefore defines the way in which the space $\mathcal{S}$ is embedded within the original $\mathbf{x}$-space. Since the mapping $\mathbf{F}_2$ can be non-linear, the sub-space $\mathcal{S}$ can be non-planar, as indicated in the figure. The mapping $\mathbf{F}_1$ then defines a projection of points in the original d-dimensional space into the M-dimensional sub-space $\mathcal{S}$.

Such a network effectively performs a non-linear principal component analysis. It has the advantage of not being limited to linear transformations, although it contains standard principal component analysis as a special case. However, the minimization of the error function is now a non-linear optimization problem, since the error function in (8.24) is no longer a quadratic function of the network parameters. Computationally intensive non-linear optimization techniques must be used (Chapter 7), and there is the risk of finding a sub-optimal local minimum of the error function. Also, the dimensionality of the sub-space must be specified in advance of training the network, so that in practice it may be necessary to train and compare several networks having different values of M. An example of the application of this approach is given in Kramer (1991).

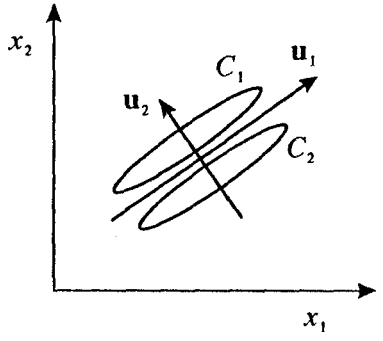

Figure 8.15. An example of a simple classification problem for which principal component analysis would discard the discriminatory information. Two-dimensional data is taken from two Gaussian classes C_1 and C_2 depicted by the two ellipses. Dimensionality reduction to one dimension using principal component analysis would give a projection of the data onto the vector $\mathbf{u}_1$ which would remove all ability to discriminate the two classes. The full discriminatory capability can be preserved if instead the data is projected onto the vector $\mathbf{u}_2$, which is the direction which would be obtained from linear discriminant analysis.

8.6.3 *Limitations of unsupervised techniques*

We have described both linear and non-linear unsupervised techniques for dimensionality reduction. These can lead to significant improvements in the performance of subsequent regression or classification systems. It should be emphasized, however, that methods based on unsupervised techniques take no account of the target data, and can therefore give results which are substantially less than optimal. A reduction in dimensionality generally involves the loss of some information, and it may happen that this information is very important for the subsequent regression or classification phase, even though it is of relatively little importance for representation of the input data itself.

As a simple example, consider a classification problem involving input data in two dimensions taken from two Gaussian-distributed classes as shown in Figure 8.15. Principal component analysis applied to this data would give the eigenvectors $\mathbf{u}_1$ and $\mathbf{u}_2$ as shown. If the dimensionality of the data were to be reduced to one dimension using principal component analysis, then the data would be projected onto the vector $\mathbf{u}_1$ since this has the larger eigenvalue. However, this would lead to a complete loss of all discriminatory information, and the classes would have identical distributions in the one-dimensional space. By contrast, a projection onto the vector $\mathbf{u}_2$ would give optimal class separation with no loss of discriminatory information. Clearly this is an extreme example, and in practice dimensionality reduction by unsupervised techniques can prove useful in many

applications.

Note that in the example of Figure 8.15, a reduction of dimensionality using Fisher's linear discriminant (Section 3.6) would yield the optimal projection vector $\mathbf{u}_2$. This is a consequence of the fact that it takes account of the class information in selecting the projection vector. However, as we saw in Section 3.6, for a problem with c classes, Fisher's linear technique can only find $c-1$ independent directions. For problems with few classes and high input dimensionality this may result in too drastic a reduction of dimensionality. Techniques such as principal component analysis do not suffer from this limitation and are able to extract any number of orthogonal directions up to the dimensionality of the original space.

It is worth noting that there is an additional link between principal component analysis and a class of linear neural network models which make use of modifications of the *Hebb learning rule* (Hebb, 1949). This form of learning involves making changes to the value of a weight parameter in proportion to the activation values of the two units which are linked by that weight. Such networks can be made to perform principal component analysis of the data (Oja, 1982, 1989; Linsker, 1988; Sanger, 1989), and furthermore it can be arranged that the weights converge to orthonormal vectors along the principal component directions. For practical applications, however, there would appear to be little advantage in using such approaches compared with standard numerical analysis techniques such as those described earlier.

8.7 Invariances and prior knowledge

Throughout this book we are considering the problem of setting up a multivariate mapping (for regression or classification) on the basis of a set of training data. In many practical situations we have, in addition to the data itself, some general information about the form which the mapping should take or some constraints which it should satisfy. This is referred to as *prior knowledge*, and its inclusion in the network design process can often lead to substantial improvements in performance.

We have already encountered one form of prior knowledge expressed as prior probabilities of class membership in a classification problem (Section 1.8). These can be taken into account in an optimal way by direct use of Bayes' theorem, or by introducing weighting factors in a sum-of-squares error function (Section 6.6.2). Here we concentrate on forms of prior knowledge concerned with various kinds of invariance. As we shall see, the required invariance properties can be built into the pre-processing stage, or they can be included in the network structure itself. While the latter option does not strictly constitute part of the pre-processing, it is discussed in this chapter for convenience.

8.7.1 *Invariances*

In many practical applications it is known that the outputs in a classification or regression problem should be unchanged, or *invariant*, when the input is subject to various transformations. An important example is the classification of objects

in two-dimensional images. A particular object should be assigned the same classification even if it is rotated or translated within the image or if it is linearly scaled (corresponding to the object moving towards or away from the camera). Such transformations produce significant changes in the raw data (expressed in terms of the intensities at each of the pixels in the image) and yet should give rise to the same output from the classification system. We shall use this object recognition example to illustrate the use of invariances in neural networks. It should be borne in mind, however, that the same general principles apply to any problem for which it is desired to incorporate invariance with respect to a set of transformations.

Broadly we can identify three basic approaches to the construction of invariant classification (or regression) systems based on neural networks (Barnard and Casasent, 1991):

1. The first approach is to train a network by example. This involves including within the training set a sufficiently large number of examples of the effects of the various transformations. Thus, for translation invariance, the training set should include examples of objects at many different positions. If suitable training data is not readily available then it can be generated by applying the transformations to the existing data, for example by translating a single image to generate several images of the same object at different locations.
2. The second approach involves making a choice of pre-processing which incorporates the required invariance properties. If features are extracted from the raw data which are themselves invariant, then any subsequent regression or classification system will necessarily also respect these invariances.
3. The final option is to build the invariance properties into the network structure itself. One way to achieve this is through the use of shared weights, and we shall consider two specific examples involving local receptive fields and higher-order networks.

While approach 1 is relatively straightforward, it suffers from the disadvantage of being inefficient in requiring a substantially expanded data set. It will also result in a network which only approximately respects the invariance. Furthermore, the network will be unable to deal with new inputs in which the range of the transformation exceeds that encountered during training, as this represents an extrapolation of the network inputs. Methods 2 and 3 achieve the required invariance properties without needing unnecessarily large data sets. In the context of translation invariance, for instance, a network which has been trained to recognize an object correctly at one position within an image can recognize the same object correctly at any position. In contrast to a network trained by method 1, such a network is able to extrapolate to new inputs if they differ from the training data primarily by virtue of one of the transformations.

An alternative approach which also involves incorporating invariances through training, but which does not require artificial expansion of the data set, is the technique of *tangent prop* (Simard *et al.*, 1992). Consider the effect of a trans-

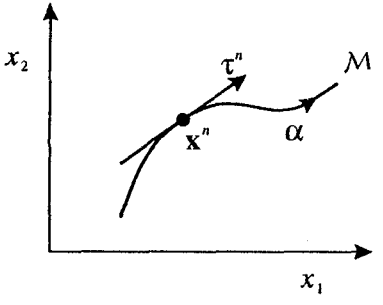

Figure 8.16. Illustration of a two-dimensional input space showing the effect of a continuous transformation on a particular input vector $\mathbf{x}^n$. A one-dimensional transformation, parametrized by the continuous variable α, applied to $\mathbf{x}^n$ causes it to sweep out a one-dimensional manifold $\mathcal{M}$. Locally, the effect of the transformation can be approximated by the tangent vector τ^n.

formation on a particular input pattern vector $\mathbf{x}^n$. Provided the transformation is continuous (such as translation or rotation, but not mirror reflection for instance) then the transformed pattern will sweep out a manifold $\mathcal{M}$ within the d-dimensional input space. This is illustrated in Figure 8.16, for the case of $d = 2$ for simplicity. Suppose the transformation is governed by a single parameter α (which might be rotation angle for instance). Then the sub-space $\mathcal{M}$ swept out by $\mathbf{x}^n$ will be one-dimensional, and will be parametrized by α. Let the vector which results from acting on $\mathbf{x}^n$ by this transformation be denoted by $\mathbf{s}(\alpha, \mathbf{x}^n)$ which is defined so that $\mathbf{s}(0, \mathbf{x}^n) = \mathbf{x}^n$. Then the tangent to the curve $\mathcal{M}$ is given by the directional derivative $\tau = \partial \mathbf{s}/\partial\alpha$, and the tangent vector at the point $\mathbf{x}^n$ is given by

$$\tau^n = \left.\frac{\partial \mathbf{s}(\alpha, \mathbf{x}^n)}{\partial \alpha}\right|_{\alpha=0}. \tag{8.25}$$

Under a transformation of the input vector, the network output vector will, in general, change. The derivative of the activation of output unit k with respect to α is given by

$$\frac{\partial y_k}{\partial \alpha} = \sum_{i=1}^{d} \frac{\partial y_k}{\partial x_i}\frac{\partial x_i}{\partial \alpha} = \sum_{i=1}^{d} J_{ki}\tau_i \tag{8.26}$$

where J_{ki} is the (k, i) element of the Jacobian matrix $\mathbf{J}$, as discussed in Section 4.9. The result (8.26) can be used to modify the standard error function, so as to encourage local invariance in the neighbourhood of the data points, by the

addition to the usual error function E of a regularization function Ω to give a total error function of the form

$$\widetilde{E} = E + \nu\Omega \tag{8.27}$$

where ν is a regularization coefficient (Section 9.2) and

$$\Omega = \frac{1}{2}\sum_n \sum_k \left(\sum_{i=1}^{d} J_{ki}^n \tau_i^n \right)^2 . \tag{8.28}$$

The regularization function will be zero when the network mapping function is invariant under the transformation in the neighbourhood of each pattern vector, and the value of the parameter ν determines the balance between the network fitting the training data and the network learning the invariance property.

In a practical implementation, the tangent vector $\boldsymbol{\tau}^n$ can be approximated by finite differences, by subtracting the original vector $\mathbf{x}^n$ from the corresponding vector after transformation using a small value of α, and dividing by α. Some smoothing of the data may also be required. The regularization function depends on the network weights through the Jacobian $\mathbf{J}$. A back-propagation formalism for computing the derivatives of the regularizer with respect to the network weights is easily obtained (Exercise 8.6) by extension of the techniques introduced in Chapter 4.

If the transformation is governed by L parameters (e.g. $L = 2$ for the case of translation in a two-dimensional image) then the space $\mathcal{M}$ will have dimensionality L, and the corresponding regularizer is given by the sum of terms of the form (8.28), one for each transformation. If several transformations are considered at the same time, and the network mapping is made invariant to each separately, then it will be (locally) invariant to combinations of the transformations (Simard *et al.*, 1992). A related technique, called *tangent distance*, can be used to build invariance properties into distance-based methods such as nearest-neighbour classifiers (Simard *et al.*, 1993).

8.7.2 *Invariance through pre-processing*

The second approach which we shall consider for incorporating invariance properties into neural network mappings is by a suitable choice of pre-processing. One such technique involves the extraction of features from the original input data which are invariant under the required transformations. Such features are often based on *moments* of the original data. For inputs which consist of a two-dimensional image, the moments are defined by

$$\iint x(u,v)K(u,v)\,du\,dv \tag{8.29}$$

where (u, v) are Cartesian coordinates describing locations within the image, $x(u, v)$ represents the intensity of the image at location (u, v), and $K(u, v)$ is called a *kernel* and is a fixed function whose form determines the particular moments under consideration. In practice, an image is specified in terms of a finite array of pixels, and so the integrals in (8.29) are replaced by discrete sums

$$\sum_i \sum_j x(u_i, v_j) K(u_i, v_j) \Delta u_i \Delta v_j. \tag{8.30}$$

When the kernel function takes the form of simple powers we have *regular moments* which, in continuous notation, can be written

$$M_{lm} = \int\int x(u, v) u^l v^m \, du \, dv \tag{8.31}$$

where l and m are non-negative integers. We can define a corresponding set of translation-invariant features, called *central moments*, by first subtracting off the means of u and v

$$\widehat{M}_{lm} = \int\int x(u, v)(u - \overline{u})^l (v - \overline{v})^m \, du \, dv \tag{8.32}$$

where $\overline{u} = M_{10}/M_{00}$ and $\overline{v} = M_{01}/M_{00}$. Under a translation of the image $x(u, v) \rightarrow x(u + \Delta u, v + \Delta v)$, and it is easy to verify that the moments defined in (8.32) are invariant. Note that this neglects edge effects and assumes that the integrals in (8.32) run over $(-\infty, \infty)$. In practice, the use of moments in the discrete form (8.30) will give only approximate invariance under such transformations.

Similarly, under a change of scale we have $x(u, v) \rightarrow x(\alpha u, \alpha v)$. We can make the central moments invariant to scale by normalizing them to give

$$\mu_{lm} = \frac{\widehat{M}_{lm}}{\widehat{M}_{00}^{1+(l+m)/2}} \tag{8.33}$$

and again it is easy to verify that the normalized moments in (8.33) are simultaneously invariant to translations and scaling. Similarly, we can use the moments in (8.33) in turn to construct moments which are simultaneously invariant to translation, scale and rotation (Exercise 8.7). For instance, the quantity

$$\mu_{20} + \mu_{02} \tag{8.34}$$

has this property (Schalkoff, 1989). Other forms of moments can also be considered which are based on different forms for the kernel function $K(u, v)$ (Khotanzad and Hong, 1990).

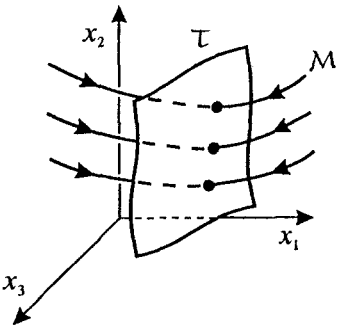

Figure 8.17. Illustration of a three-dimensional input space showing trajectories, such as $\mathcal{M}$, which patterns sweep out under the action of transformations to which the network outputs should be invariant. A suitably chosen set of constraints will define a sub-space $\mathcal{T}$ which intersects each trajectory precisely once. If new inputs are mapped onto this surface using the transformations then invariance is guaranteed.

One problem with the use of moments as input features is that considerable computational effort may be required for their evaluation, and this computation must be repeated for each new input image. A second problem is that a lot of information is discarded in evaluating any particular moment, and so many moments may be required in order to give good discrimination.

An alternative, related approach to invariant pre-processing is to transform any new inputs so as to satisfy some appropriately chosen set of constraints (Barnard and Casasent, 1991). This is illustrated schematically in Figure 8.17 for a set of one-parameter transformations. Under the action of the transformations, each input vector sweeps out a trajectory $\mathcal{M}$ as discussed earlier. Those patterns which satisfy the constraints live on a sub-space $\mathcal{T}$ which intersects the trajectories. Note that the constraints must be chosen so that each trajectory intersects the constraint surface at precisely one point. Any new input vector is first transformed (thus moving it along its trajectory) until it reaches the constraint surface. This transformed vector is then used as the input to the network. As an example, suppose we wish to impose invariance to translations and changes of scale. The constraints might then take the form that the zeroth and first moments M_{00}, M_{10} and M_{01}, given by (8.31), should have specified values. Every image (for the training set or test set) is first transformed by translation and scaling until the constraints are satisfied.

8.7.3 *Shared weights*

The third approach to dealing with invariances, discussed above, involves structuring the network itself in such a way that the network mapping respects the

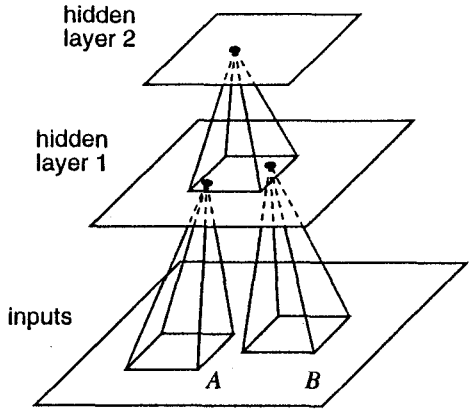

Figure 8.18. Schematic architecture of a network for translation-invariant object recognition in two-dimensional images. In a practical system there may be more than two layers between the input image and the outputs.

invariances. While, strictly, this is not a form of pre-processing, it is treated here for convenience. Again, we introduce this concept in the context of networks designed for object recognition in two-dimensional images.

Consider the network structure shown in Figure 8.18. The inputs to the network are given by the intensities at each of the pixels in a two-dimensional array. Units in the first and second layers are similarly arranged in two-dimensional sheets to reflect the geometrical structure of the problem. Instead of having full interconnections between adjacent layers, each hidden unit receives inputs only from units in a small region in the previous layer, known as a *receptive field.* This reflects the results of experiments in conventional image processing which have demonstrated the advantage of extracting local features from an image and then combining them together to form higher-order features. Note that it also imitates some aspects of the mammalian visual processing system. The network architecture is typically chosen so that there is some overlap between adjacent receptive fields.

The technique of *shared weights* can then be used to build in some degree of translation invariance into the response of the network (Rumelhart *et al.*, 1986; Le Cun *et al.*, 1989; Lang *et al.*, 1990). In the simplest case this involves constraining the weights from each receptive field to be equal to the corresponding weights from all of the receptive fields of the other units in the same layer. Consider an object which falls within receptive field shown at A in Figure 8.18, corresponding to a unit in hidden layer 1, and which produces some activation level in that unit. If the same object falls at the corresponding position in receptive field B, then, as a consequence of the shared weights, the corresponding

unit in hidden layer 1 will have the same activation level. The units in the second layer have fixed weights chosen so that each unit computes a simple average of the activations of the units that fall within its receptive field. This allows units in the second layer to be relatively insensitive to moderate translations within the input image. However, it does preserve some positional information thereby allowing units in higher layers to detect more complex composite features. Typically each successive layer has fewer units than previous layers, as information on the spatial location of objects is gradually eliminated. This corresponds to the use of a relatively high resolution to detect the presence of a feature in an earlier layer, while using a lower resolution to represent the location of that feature in a subsequent layer.

In a practical network there may be several pairs of layers, with alternate layers having fixed and adaptive weights. These gradually build up increasing tolerance to shifts in the input image, so that the final output layer has a response which is almost entirely independent of the position of an object in the input field.

As described so far, this network architecture has only one kind of receptive field in each layer. In order to be able to extract several different kinds of feature is necessary to provide several 'planes' of units in each hidden layer, with all units in a given plane sharing the same weights. Weight sharing can be enforced during learning by initializing corresponding weights to the same (random) values and then averaging the weight changes for all of the weights in one group and updating all of the corresponding weights by the same amount using the averaged weight change.

Network architectures of this form have been used in the zip code recognition system of Le Cun *et al.* (1989), and in the *neocognitron* of Fukushima *et al.* (1983) and Fukushima (1988), for translation-invariant recognition of handwritten digits.

The use of receptive fields can dramatically reduce the number of weights present in the network compared with a fully connected architecture. This makes it practical to treat pixel values in an image directly as inputs to a network. In addition, the use of shared weights means that the number of independent parameters in the network is much less than the number of weights, which allows much smaller data sets to be used than would otherwise be necessary.

8.7.4 *Higher-order networks for encoding invariances*

In Section 4.5 we introduced the concept of a higher-order network based on units whose outputs are given by

$$z_j = g\left(w_j + \sum_{i_1=1}^{d} w_{ji_1} x_{i_1} + \sum_{i_1=1}^{d}\sum_{i_2=1}^{d} w_{ji_1i_2} x_{i_1} x_{i_2} + \cdots\right) \tag{8.35}$$

where x_i is an input, $g(\cdot)$ is a non-linear activation function and the w's represent the weights. We have already remarked that such networks can have a

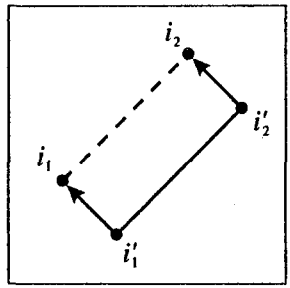

Figure 8.19. We can impose translation invariance on a second-order network if we ensure that, for each hidden unit separately, weights from any pair of points i_1 and i_2 are constrained to equal those from any other pair i_1' and i_2', where the line i_1'–i_2' can be obtained from the line i_1–i_2 by translation.

proliferation of weight parameters and are therefore impractical for many applications. (The number of independent parameters per unit is the same as for the corresponding multivariate polynomial, and is discussed in Exercises 1.6–1.8.) However, we can exploit the structure of a higher-order network to impose invariances, and at the same time reduce significantly the number of independent weights in the network, by using a form of weight sharing (Giles and Maxwell, 1987; Perantonis and Lisboa, 1992). Consider the problem of incorporating translation invariance into a higher-order network. This can be achieved by using a second-order network of the form

$$z_j = g\left(\sum_{i_1}\sum_{i_2} w_{ji_1i_2} x_{i_1} x_{i_2}\right). \tag{8.36}$$

Under a translation, the value of the intensity in pixel i_1 will go from its original value x_{i_1} to a new value x'_{i_1} given by $x'_{i_1} = x_{i_1'}$ where the translation can be described by a vector from pixel $i_1{}'$ to pixel i_1. Thus the argument of the activation function $g(\cdot)$ in (8.36) will be invariant if, for each unit j in the first hidden layer, we have

$$w_{ji_1i_2} = w_{ji_1'i_2'}. \tag{8.37}$$

This has a simple geometrical interpretation as indicated in Figure 8.19. Each unit in the first hidden layer takes inputs from two pixels in the image, such as those labelled i_1 and i_2 in the figure. The constraint in (8.37) requires that, for each unit in the first hidden layer, and for each possible pair of points in the image, the weights from any other pair of points, such as those at i_1' and i_2' which can be obtained from i_1 and i_2 by translation, must be equal. Note that such

an approach would not work with a first-order network, since the constraint on the weights would force all weights into any given unit to be equal. Each unit would therefore take as input something proportional to the average of all of the input pixel values and, while this would be translation invariant, there would be no freedom left for the units to detect any structure in the image. Edge effects, as well as the discrete nature of the pixels, have been neglected here, and in practice the invariance properties will be only approximately realized.

Higher-order networks can be made invariant to more complex transformations. Consider a general Kth-order unit

$$\sum_{i_1} \cdots \sum_{i_K} w_{ji_1,\cdots,i_K} x_{i_1}, \cdots, x_{i_K}. \tag{8.38}$$

Under a particular geometrical transformation, $x_{i_l} \rightarrow x'_{i_l} = x_{i'_l}$ where the pixel at i_l is replaced by the pixel at i'_l. It follows that the expression in (8.38) will be invariant provided

$$w_{ji'_1,\cdots,i'_K} = w_{ji_1,\cdots,i_K}. \tag{8.39}$$

As well as allowing invariances to be built into the network structure, the imposition of the constraints in (8.39) can greatly reduce the number of free parameters in the network, and thereby dramatically reduce the size of data set needed to determine those weights.

Simultaneous translation and scale invariance can be built into a second-order network by demanding that, for each unit in the first hidden layer, and for each pair of inputs i_1 and i_2, the weights from i_1 and i_2 are constrained to equal those from any other pair i'_1 and i'_2 where the pair i'_1–i'_2 can be obtained from i_1–i_2 by a combination of translation and scaling. This selects all pairs of points such that the line i'_1–i'_2 is parallel to the line i_1–i_2. There is a slight complication in the case of scaling arising from the fact that the input image consists of discrete pixels. If a given geometrical object is scaled by a factor λ then the number of pixels which it occupies is scaled by a factor λ^2. If the image consists of black pixels (value +1) on a white background (value 0) for instance, then the number of active pixels will be scaled by λ^2, which would spoil the scale invariance. The problem can be avoided by normalizing the image, e.g. to a vector of unit length. Note that this then gives fractional values for the inputs.

If we consider simultaneous translation, rotation and scale invariance, we see that any pair of points can be mapped to any other pair by a combination of such transformations. Thus a second-order network would be constrained to have all weights to any hidden unit equal, which would again cause the activation of each unit to be simply proportional to the average of the input values. We therefore need to go to a third-order network. In this case, each unit takes inputs from three pixels in the image, and the weights must satisfy the constraint that, for every triplet of pixels, and for every hidden unit, the weights must equal those

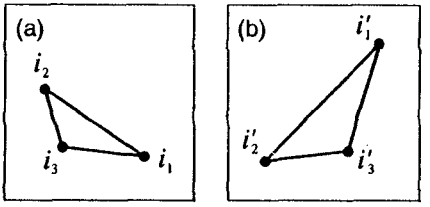

Figure 8.20. Simultaneous translation, rotation and scale invariance can be built into a third-order network provided weights from triplets of points which correspond to similar triangles, such as those shown in (a) and (b), are constrained to be equal.

emanating from any other triplet which can be obtained by any combination of translations, rotations and scalings (Reid *et al.*, 1989). This means that corresponding triplets lie at the vertices of *similar* triangles, in other words triangles which have the same values of the angles encountered in the same order when traversing the triangle in, say, a clockwise direction. This is illustrated in Figure 8.20. Although the incorporation of constraints greatly reduces the number of free parameters in higher-order networks, the use of such networks is not widespread.

Exercises

8.1 ($\star$) Verify that the whitened input vector, given by (8.6), has zero mean and a covariance matrix given by the identity matrix.

8.2 ($\star$) Consider a radial basis function network with spherical Gaussian basis functions in which the jth basis function is governed by a mean $\boldsymbol{\mu}_j$ and a variance parameter σ_j^2 (Section 5.2). Show that the effect of applying the whitening transformation (8.6) to the original input data is equivalent to a special case of the same network with general Gaussian basis functions governed by a general covariance matrix $\widetilde{\boldsymbol{\Sigma}}_j$ in which the original un-whitened data is used. Obtain an expression for the corresponding mean $\widetilde{\boldsymbol{\mu}}_j$ and covariance matrix $\widetilde{\boldsymbol{\Sigma}}_j$ in terms of the parameters of the original basis functions and of the whitening transformation.

8.3 ($\star\star$) Generate sets of data points in two dimensions using a variety of distributions including Gaussian (with general covariance matrix) and mixtures of Gaussians. For each data set, apply the whitening transformation (Section 8.2) and produce scatter plots of the data points before and after transformation.

8.4 ($\star$) Consider a trained classifier which can produce the posterior probabilities $P(\mathcal{C}_k|\mathbf{x})$ for a new input vector $\mathbf{x}$. Suppose that some of the values of the input vector are missing, so that $\mathbf{x}$ can be partitioned into a subvector $\mathbf{x}_m$ of components whose values are missing, and a remaining vector

$\hat{\mathbf{x}}$ whose values are present. Show that posterior probabilities, given only the data $\hat{\mathbf{x}}$, are given by

$$P(\mathcal{C}_k|\hat{\mathbf{x}}) = \frac{1}{p(\hat{\mathbf{x}})} \int P(\mathcal{C}_k|\hat{\mathbf{x}}, \mathbf{x}_m) p(\hat{\mathbf{x}}, \mathbf{x}_m)\, d\mathbf{x}_m. \tag{8.40}$$

8.5 ($\star$) Consider the problem of selecting M feature variables from a total of d candidate variables. Find expressions for the number of criterion function evaluations which must be performed for (i) exhaustive search, (ii) sequential forward selection, and (iii) sequential backward elimination. Consider the case of choosing 10 features out of a set of 50 candidates, and evaluate the corresponding expressions for the number of evaluations by these three methods.

8.6 ($\star\star$) Consider a multi-layer perceptron with arbitrary feed-forward topology, which is to be trained by minimizing the 'tangent prop' error function (8.27) in which the regularizing function is given by (8.28). Show that the regularization term Ω can be written as a sum over patterns of terms of the form

$$\Omega^n = \frac{1}{2} \sum_k (\mathcal{D} y_k)^2 \tag{8.41}$$

where $\mathcal{D}$ is a differential operator defined by

$$\mathcal{D} \equiv \sum_i \tau_i \frac{\partial}{\partial x_i}. \tag{8.42}$$

By acting on the forward propagation equations

$$z_j = g(a_j), \qquad a_j = \sum_i w_{ji} z_i \tag{8.43}$$

with the operator $\mathcal{D}$, show that Ω^n can be evaluated by forward propagation using the following equations:

$$\xi_j = g'(a_j)\alpha_j, \qquad \alpha_j = \sum_i w_{ji} \xi_i. \tag{8.44}$$

where we have defined the new variables

$$\xi_j \equiv \mathcal{D} z_j, \qquad \alpha_j \equiv \mathcal{D} a_j. \tag{8.45}$$

Now show that the derivatives of Ω^n with respect to a weight w_{rs} in the network can be written in the form

$$\frac{\partial \Omega^n}{\partial w_{rs}} = \sum_k \xi_k \left\{ \phi_r^k z_s + \delta_r^k \xi_s \right\} \tag{8.46}$$

where we have defined

$$\delta_r^k \equiv \frac{\partial y_k}{\partial a_r}, \qquad \phi_r^k \equiv \mathcal{D}\delta_r^k. \tag{8.47}$$

Write down the back-propagation equations for δ_r^k, and hence derive a set of back-propagation equations for the evaluation of the ϕ_r^k.

8.7 ($\star$) We have seen that the normalized moments μ_{lm} defined by (8.33) are simultaneously invariant to translation and scaling. It follows that any combination of such moments will also satisfy the same invariances. Show that the moment defined in (8.34) is, additionally, invariant under rotation $\theta \to \theta + \Delta\theta$. Hint: this is most easily done by representing the moments using polar coordinates centred on the point $(\overline{u}, \overline{v})$, so that the central moments become

$$\widehat{M}_{lm} = \int\int x(r,\theta)(r\cos\theta)^l(r\sin\theta)^m r\, dr\, d\theta, \tag{8.48}$$

and then making use of the relation $\sin^2\theta + \cos^2\theta = 1$. Which of the following moments are rotation invariant?

(a) $$(\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2 \tag{8.49}$$

(b) $$(\mu_{20} + \mu_{02})^2 - 4\mu_{11}^2 \tag{8.50}$$

(c) $$(\mu_{30} + 3\mu_{12})^2 - (3\mu_{21} + \mu_{03})^2 \tag{8.51}$$

(d) $$(\mu_{30} - 3\mu_{12})^2 + (3\mu_{21} - \mu_{03})^2. \tag{8.52}$$