



# Neural Networks for Pattern Recognition

Christopher M. Bishop

## STATISTICAL PATTERN RECOGNITION

The term pattern recognition encompasses a wide range of information processing problems of great practical significance, from speech recognition and the classification of handwritten characters, to fault detection in machinery and medical diagnosis. Often these are problems which many humans solve in a seemingly effortless fashion. However, their solution using computers has, in many cases, proved to be immensely difficult. In order to have the best opportunity of developing effective solutions, it is important to adopt a principled approach based on sound theoretical concepts.

The most general, and most natural, framework in which to formulate solutions to pattern recognition problems is a statistical one, which recognizes the probabilistic nature both of the information we seek to process, and of the form in which we should express the results. Statistical pattern recognition is a well established field with a long history. Throughout this book, we shall view neural networks as an extension of conventional techniques in statistical pattern recognition, and we shall build on, rather than ignore, the many powerful results which this field offers.

In this first chapter we provide a gentle introduction to many of the key concepts in pattern recognition which will be central to our treatment of neural networks. By using a simple pattern classification example, and analogies to the problem of curve fitting, we introduce a number of important issues which will re-emerge in later chapters in the context of neural networks. This chapter also serves to introduce some of the basic formalism of statistical pattern recognition.

### 1.1 An example – character recognition

We can introduce many of the fundamental concepts of statistical pattern recognition by considering a simple, hypothetical, problem of distinguishing handwritten versions of the characters ‘a’ and ‘b’. Images of the characters might be captured by a television camera and fed to a computer, and we seek an algorithm which can distinguish as reliably as possible between the two characters. An image is represented by an array of pixels, as illustrated in Figure 1.1, each of which carries an associated value which we shall denote by  $x_i$  (where the index  $i$  labels the individual pixels). The value of  $x_i$  might, for instance, range from 0 for a completely white pixel to 1 for a completely black pixel. It is often convenient to gather the  $x_i$  variables together and denote them by a single vector  $\mathbf{x} = (x_1, \dots, x_d)^T$  where  $d$  is the total number of such variables, and the

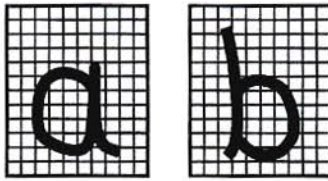


Figure 1.1. Illustration of two hypothetical images representing handwritten versions of the characters ‘a’ and ‘b’. Each image is described by an array of pixel values  $x_i$  which range from 0 to 1 according to the fraction of the pixel square occupied by black ink.

superscript T denotes the transpose. In considering this example we shall ignore a number of detailed practical considerations which would have to be addressed in a real implementation, and focus instead on the underlying issues.

The goal in this classification problem is to develop an algorithm which will assign any image, represented by a vector  $\mathbf{x}$ , to one of two classes, which we shall denote by  $C_k$ , where  $k = 1, 2$ , so that class  $C_1$  corresponds to the character ‘a’ and class  $C_2$  corresponds to ‘b’. We shall suppose that we are provided with a large number of examples of images corresponding to both ‘a’ and ‘b’, which have already been classified by a human. Such a collection will be referred to as a *data set*. In the statistics literature it would be called a *sample*.

One obvious problem which we face stems from the high dimensionality of the data which we are collecting. For a typical image size of  $256 \times 256$  pixels, each image can be represented as a point in a  $d$ -dimensional space, where  $d = 65\,536$ . The axes of this space represent the grey-level values of the corresponding pixels, which in this example might be represented by 8-bit numbers. In principle we might think of storing every possible image together with its corresponding class label. In practice, of course, this is completely impractical due to the very large number of possible images: for a  $256 \times 256$  image with 8-bit pixel values there would be  $2^{8 \times 256 \times 256} \simeq 10^{158\,000}$  different images. By contrast, we might typically have a few thousand examples in our training set. It is clear then that the classifier system must be designed so as to be able to classify correctly a previously unseen image vector. This is the problem of *generalization*, which is discussed at length in Chapters 9 and 10.

As we shall see in Section 1.4, the presence of a large number of input variables can present some severe problems for pattern recognition systems. One technique to help alleviate such problems is to combine input variables together to make a smaller number of new variables called *features*. These might be constructed ‘by hand’ based on some understanding of the particular problem being tackled, or they might be derived from the data by automated procedures. In the present example, we could, for instance, evaluate the ratio of the height of the character to its width, which we shall denote by  $\tilde{x}_1$ , since we might expect that characters

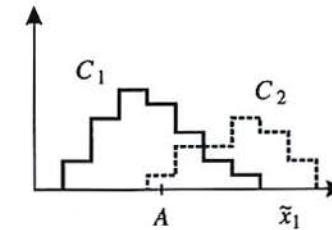


Figure 1.2. Schematic plot of the histograms of the feature variable  $\tilde{x}_1$  given by the ratio of the height of a character to its width, for a data set of images containing examples from classes  $C_1 \equiv$  ‘a’ and  $C_2 \equiv$  ‘b’. Notice that characters from class  $C_2$  tend to have larger values of  $\tilde{x}_1$  than characters from class  $C_1$ , but that there is a significant overlap between the two histograms. If a new image is observed which has a value of  $\tilde{x}_1$  given by  $A$ , we might expect the image is more likely to belong to class  $C_1$  than  $C_2$ .

from class  $C_2$  (corresponding to ‘b’) will typically have larger values of  $\tilde{x}_1$  than characters from class  $C_1$  (corresponding to ‘a’). We might then hope that the value of  $\tilde{x}_1$  alone will allow new images to be assigned to the correct class. Suppose we measure the value of  $\tilde{x}_1$  for each of the images in our data set, and plot their values as histograms for each of the two classes. Figure 1.2 shows the form which these histograms might take. We see that typically examples of the character ‘b’ have larger values of  $\tilde{x}_1$  than examples of the character ‘a’, but we also see that the two histograms overlap, so that occasionally we might encounter an example of ‘b’ which has a smaller value of  $\tilde{x}_1$  than some example of ‘a’. We therefore cannot distinguish the two classes perfectly using the value of  $\tilde{x}_1$  alone.

If we suppose for the moment that the only information available is the value of  $\tilde{x}_1$ , we may wish to know how to make best use of it to classify a new image so as to minimize the number of misclassifications. For a new image which has a value of  $\tilde{x}_1$  given by  $A$  as indicated in Figure 1.2, we might expect that the image is more likely to belong to class  $C_1$  than to class  $C_2$ . One approach would therefore be to build a classifier system which simply uses a threshold for the value of  $\tilde{x}_1$  and which classifies as  $C_2$  any image for which  $\tilde{x}_1$  exceeds the threshold, and which classifies all other images as  $C_1$ . We might expect that the number of misclassifications in this approach would be minimized if we choose the threshold to be at the point where the two histograms cross. This intuition turns out to be essentially correct, as we shall see in Section 1.9.

The classification procedure we have described so far is based on the evaluation of  $\tilde{x}_1$  followed by its comparison with a threshold. While we would expect this to give some degree of discrimination between the two classes, it suffers from the problem, indicated in Figure 1.2, that there is still significant overlap of the histograms, and hence we must expect that many of the new characters on which we might test it will be misclassified. One way to try to improve the

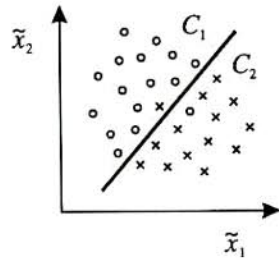


Figure 1.3. A hypothetical classification problem involving two feature variables  $\tilde{x}_1$  and  $\tilde{x}_2$ . Circles denote patterns from class  $C_1$  and crosses denote patterns from class  $C_2$ . The decision boundary (shown by the line) is able to provide good separation of the two classes, although there are still a few patterns which would be incorrectly classified by this boundary. Note that if the value of either of the two features were considered separately (corresponding to a projection of the data onto one or other of the axes), then there would be substantially greater overlap of the two classes.

situation is to consider a second feature  $\tilde{x}_2$  (whose actual definition we need not consider) and to try to classify new images on the basis of the values of  $\tilde{x}_1$  and  $\tilde{x}_2$  considered together. The reason why this might be beneficial is indicated in Figure 1.3. Here we see examples of patterns from two classes plotted in the  $(\tilde{x}_1, \tilde{x}_2)$  space. It is possible to draw a line in this space, known as a *decision boundary*, which gives good separation of the two classes. New patterns which lie above the decision boundary are classified as belonging to class  $C_1$  while patterns falling below the decision boundary are classified as  $C_2$ . A few examples are still incorrectly classified, but the separation of the patterns is much better than if either feature had been considered individually, as can be seen by considering all of the data points projected as histograms onto one or other of the two axes.

We could continue to consider ever larger numbers of (independent) features in the hope of improving the performance indefinitely. In fact, as we shall see in Section 1.4, adding too many features can, paradoxically, lead to a worsening of performance. Furthermore, for many real pattern recognition applications, it is the case that some overlap between the distributions of the classes is inevitable. This highlights the intrinsically probabilistic nature of the pattern classification problem. With handwritten characters, for example, there is considerable variability in the way the characters are drawn. We are forced to treat the measured variables as random quantities, and to accept that perfect classification of new examples may not always be possible. Instead we could aim to build a classifier which has the smallest probability of making a mistake.

## 1.2 Classification and regression

The system considered above for classifying handwritten characters was designed to take an image and to assign it to one of the two classes  $C_1$  or  $C_2$ . We can represent the outcome of the classification in terms of a variable  $y$  which takes the value 1 if the image is classified as  $C_1$ , and the value 0 if it is classified as  $C_2$ . Thus, the overall system can be viewed as a mapping from a set of input variables  $x_1, \dots, x_d$ , representing the pixel intensities, to an output variable  $y$  representing the class label. In more complex problems there may be several output variables, which we shall denote by  $y_k$  where  $k = 1, \dots, c$ . Thus, if we wanted to classify all 26 letters of the alphabet, we might consider 26 output variables each of which corresponds to one of the possible letters.

In general it will not be possible to determine a suitable form for the required mapping, except with the help of a data set of examples. The mapping is therefore modelled in terms of some mathematical function which contains a number of adjustable parameters, whose values are determined with the help of the data. We can write such functions in the form

$$y_k = y_k(\mathbf{x}; \mathbf{w}) \quad (1.1)$$

where  $\mathbf{w}$  denotes the vector of parameters. A neural network model, of the kind considered in this book, can be regarded simply as a particular choice for the set of functions  $y_k(\mathbf{x}; \mathbf{w})$ . In this case, the parameters comprising  $\mathbf{w}$  are often called *weights*. For the character classification example considered above, the threshold on  $x$  was an example of a parameter whose value was found from the data by plotting histograms as in Figure 1.2. The use of a simple threshold function, however, corresponds to a very limited form for  $y(\mathbf{x}; \mathbf{w})$ , and for most practical applications we need to consider much more flexible functions. The importance of neural networks in this context is that they offer a very powerful and very general framework for representing non-linear mappings from several input variables to several output variables, where the form of the mapping is governed by a number of adjustable parameters. The process of determining the values for these parameters on the basis of the data set is called *learning* or *training*, and for this reason the data set of examples is generally referred to as a *training set*. Neural network models, as well as many conventional approaches to statistical pattern recognition, can be viewed as specific choices for the functional forms used to represent the mapping (1.1), together with particular procedures for optimizing the parameters in the mapping. In fact, neural network models often contain conventional approaches as special cases, as discussed in subsequent chapters.

In classification problems the task is to assign new inputs to one of a number of discrete classes or categories. However, there are many other pattern recognition tasks, which we shall refer to as *regression* problems, in which the outputs represent the values of continuous variables. Examples include the determination of the fraction of oil in a pipeline from measurements of the attenuation

of gamma beams passing through the pipe, and the prediction of the value of a currency exchange rate at the some future time, given its values at a number of recent times. In fact, as discussed in Section 2.4, the term ‘regression’ refers to a specific kind of function defined in terms of an average over a random quantity. Both regression and classification problems can be seen as particular cases of *function approximation*. In the case of regression problems it is the regression function (defined in Section 6.1.3) which we wish to approximate, while for classification problems the functions which we seek to approximate are the probabilities of membership of the different classes expressed as functions of the input variables. Many of the key issues which need to be addressed in tackling pattern recognition problems are common both to classification and regression.

### 1.3 Pre-processing and feature extraction

Rather than represent the entire transformation from the set of input variables  $x_1, \dots, x_d$  to the set of output variables  $y_1, \dots, y_c$  by a single neural network function, there is often great benefit in breaking down the mapping into an initial *pre-processing* stage, followed by the parametrized neural network model itself. This is illustrated schematically in Figure 1.4. For many applications, the outputs from the network also undergo *post-processing* to convert them to the required form. In our character recognition example, the original input variables, given by the pixel values  $x_i$ , were first transformed to a single variable  $\tilde{x}_1$ . This is an example of a form of pre-processing which is generally called *feature extraction*. The distinction between the pre-processing stage and the neural network is not always clear cut, but often the pre-processing can be regarded as a fixed transformation of the variables, while the network itself contains adaptive parameters whose values are set as part of the training process. The use of pre-processing can often greatly improve the performance of a pattern recognition system, and there are several reasons why this may be so, as we now discuss.

In our character recognition example, we know that the decision on whether to classify a character as ‘a’ or ‘b’ should not depend on where in the image that character is located. A classification system whose decisions are insensitive to the location of an object within an image is said to exhibit *translation invariance*. The simple approach to character recognition considered above satisfies this property because the feature  $\tilde{x}_1$  (the ratio of height to width of the character) does not depend on the character’s position. Note that this feature variable also exhibits *scale invariance*, since it is unchanged if the size of the character is uniformly re-scaled. Such invariance properties are examples of *prior knowledge*, that is, information which we possess about the desired form of the solution which is additional to the information provided by the training data. The inclusion of prior knowledge into the design of a pattern recognition system can improve its performance dramatically, and the use of pre-processing is one important way of achieving this. Since pre-processing and feature extraction can have such a significant impact on the final performance of a pattern recognition system, we have devoted the whole of Chapter 8 to a detailed discussion of these topics.

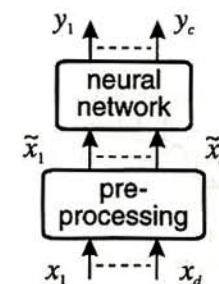


Figure 1.4. The majority of neural network applications require the original input variables  $x_1, \dots, x_d$  to be transformed by some form of pre-processing to give a new set of variables  $\tilde{x}_1, \dots, \tilde{x}_d$ . These are then treated as the inputs to the neural network, whose outputs are denoted by  $y_1, \dots, y_c$ .

### 1.4 The curse of dimensionality

There is another important reason why pre-processing can have a profound effect on the performance of a pattern recognition system. To see this let us return again to the character recognition problem, where we saw that increasing the number of features from 1 to 2 could lead to an improvement in performance. This suggests that we might use an ever larger number of such features, or even dispense with feature extraction altogether and simply use all 65 536 pixel values directly as inputs to our neural network. In practice, however, we often find that, beyond a certain point, adding new features can actually lead to a *reduction* in the performance of the classification system. In order to understand this important effect, consider the following very simple technique (not recommended in practice) for modelling non-linear mappings from a set of input variables  $x_i$  to an output variable  $y$  on the basis of a set of training data.

We begin by dividing each of the input variables into a number of intervals, so that the value of a variable can be specified approximately by saying in which interval it lies. This leads to a division of the whole input space into a large number of boxes or cells as indicated in Figure 1.5. Each of the training examples corresponds to a point in one of the cells, and carries an associated value of the output variable  $y$ . If we are given a new point in the input space, we can determine a corresponding value for  $y$  by finding which cell the point falls in, and then returning the average value of  $y$  for all of the training points which lie in that cell. By increasing the number of divisions along each axis we could increase the precision with which the input variables can be specified. There is, however, a major problem. If each input variable is divided into  $M$  divisions, then the total number of cells is  $M^d$  and this grows *exponentially* with the dimensionality of the input space. Since each cell must contain at least one data point, this implies that the quantity of training data needed to specify the mapping also grows exponentially. This phenomenon has been termed the *curse of dimensionality*.