1
Introduction

1.1 Supervised Learning

This book is about the use of artificial neural networks for supervised learning problems. Many such problems occur in practical applications of artificial neural networks. For example, a neural network might be used as a component of a face recognition system for a security application. After seeing a number of images of legitimate users' faces, the network needs to determine accurately whether a new image corresponds to the face of a legitimate user or an imposter. In other applications, such as the prediction of future price of shares on the stock exchange, we may require a neural network to model the relationship between a pattern and a real-valued quantity.

In general, in a supervised learning problem, the learning system must predict the *labels* of patterns, where the label might be a class label or a real number. During *training*, it receives some partial information about the true relationship between patterns and their labels in the form of a number of correctly labelled patterns. For example, in the face recognition application, the learning system receives a number of images, each labelled as either a legitimate user or an imposter. Learning to accurately label patterns from training data in this way has two major advantages over designing a hard-wired system to solve the same problem: it can save an enormous amount of design effort, and it can be used for problems that cannot easily be specified precisely in advance, perhaps because the environment is changing.

In designing a learning system for a supervised learning problem, there are three key questions that must be considered. The first of these concerns *approximation*, or representational, properties: we can associate with a learning system the class of mappings between patterns and labels

that it can produce, but is this class sufficiently powerful to approximate accurately enough the true relationship between the patterns and their labels? The second key issue is a statistical one concerning *estimation*: since we do not know the true relationship between patterns and their labels, and instead receive only a finite amount of data about this relationship, how much data suffices to model the relationship with the desired accuracy? The third key question is concerned with the computational efficiency of learning algorithms: how can we *efficiently* make use of the training data to choose an accurate model of the relationship?

In this book, we concentrate mainly on the estimation question, although we also investigate the issues of computation and, to a lesser extent, approximation. Many of the results are applicable to a large family of function classes, but we focus on artificial neural networks.

1.2 Artificial Neural Networks

Artificial neural networks have become popular over the last ten years for diverse applications from financial prediction to machine vision. Although these networks were originally proposed as simplified models of biological neural networks, we are concerned here with their application to supervised learning problems. Consequently, we omit the word 'artificial,' and we consider a neural network as nothing more than a certain type of nonlinear function. In this section we introduce two of the neural network classes that are discussed later in the book and use them to illustrate the key issues of approximation, estimation, and computation described above.

The simple perceptron

First we consider the *simple (real-input) perceptron*, which computes a function from $\mathbb{R}^n$ to $\{0,1\}$. Networks such as this, whose output is either 0 or 1, are potentially suitable for pattern classification problems in which we wish to divide the patterns into two classes, labelled '0' and '1'. A simple perceptron computes a function f of the form

$$f(x) = \operatorname{sgn}(w \cdot x - \theta),$$

for input vector $x \in \mathbb{R}^n$, where $w = (w_1, \ldots, w_n) \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ are adjustable parameters, or *weights* (the particular weight θ being known

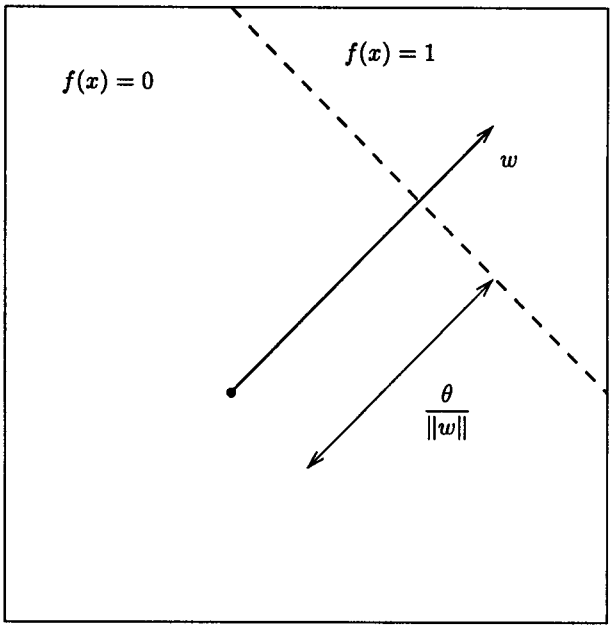

Fig. 1.1. The decision boundary in $\mathbb{R}^2$ computed by a simple perceptron with parameters w, θ.

as the *threshold*). Here, $w \cdot x$ denotes the inner product $\sum_{i=1}^{n} w_i x_i$, and

$$\text{sgn}(\alpha) = \begin{cases} 1 & \text{if } \alpha \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, the decision boundary of this function (that is, the boundary between the set of points classified as 0 and those classified as 1) is the affine subspace of $\mathbb{R}^n$ defined by the equation $w \cdot x - \theta = 0$. Figure 1.1 shows an example of such a decision boundary. Notice that the vector w determines the orientation of the boundary, and the ratio $\theta/\|w\|$ determines its distance from the origin (where $\|w\| = \left(\sum_{i=1}^{n} w_i^2\right)^{1/2}$).

Suppose we wish to use a simple perceptron for a pattern classification problem, and that we are given a collection of labelled data ((x, y) pairs) that we want to use to find good values of the parameters w and θ. The *perceptron algorithm* is a suitable method. This algorithm starts with arbitrary values of the parameters, and cycles through the training data, updating the parameters whenever the perceptron misclassifies an example. If the current function f misclassifies the pair (x, y) (with

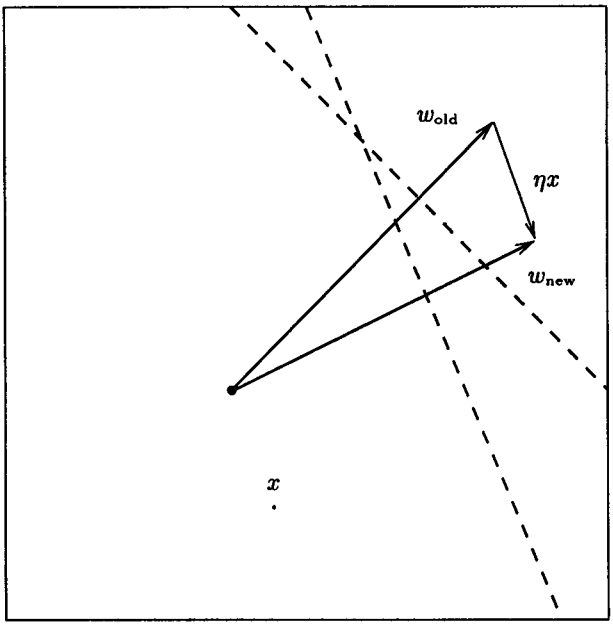

Fig. 1.2. The perceptron algorithm updates the parameters to move the decision boundary towards a misclassified example.

$x \in \mathbb{R}^n$ and $y \in \{0,1\}$), the algorithm adds $\eta(y - f(x))x$ to w and $\eta(f(x) - y)$ to θ, where η is a (prescribed) fixed positive constant. This update has the effect of moving the decision boundary closer to the misclassified point x (see Figure 1.2).

As we shall see in Chapter 24, after a finite number of iterations this algorithm finds values of the parameters that correctly classify all of the training examples, provided such parameters exist.

It is instructive to consider the key issues of approximation, estimation, and computation for the simple perceptron. Although we shall not study its approximation capabilities in this book, we mention that the representational capabilities of the simple perceptron are rather limited. This is demonstrated, for instance, by the fact that for binary input variables ($x \in \{0,1\}^n$), the class of functions computed by the perceptron forms a tiny fraction of the total number of boolean functions. Results in the first two parts of this book provide answers to the estimation question for classes of functions such as simple perceptrons. It might not suffice simply to find parameter values that give correct classifications

for all of the training examples, since we would also like the perceptron to perform well on subsequent (as yet unseen) data. We are led to the problem of *generalization*, in which we ask how the performance on the training data relates to subsequent performance. In the next chapter, we describe some assumptions about the process generating the training data (and subsequent patterns), which will allow us to pose such questions more precisely. In the last part of the book, we study the computation question for a number of neural network function classes. Whenever it is possible, the perceptron algorithm is guaranteed to find, in a finite number of iterations, parameters that correctly classify all of the training examples. However, it is desirable that the number of iterations required does not grow too rapidly as a function of the problem complexity (measured by the input dimension and the training set size). Additionally, if there are no parameter values that classify all of the training set correctly, we should like a learning algorithm to find a function that minimizes the number of mistakes made on the training data. In general the perceptron algorithm will not converge to such a function. Indeed, as we shall see, it is known that no algorithm can efficiently solve this problem (given standard complexity theoretic assumptions).

The two-layer real-output sigmoid network

As a second example, we now consider the *two-layer real-output sigmoid network*. This network computes a function f from $\mathbb{R}^n$ to $\mathbb{R}$ of the form

$$f(x) = \sum_{i=1}^{k} w_i \sigma\left(v_i \cdot x + v_{i,0}\right) + w_0,$$

where $x \in \mathbb{R}^n$ is the input vector, $w_i \in \mathbb{R}$ $(i = 0, \ldots, k)$ are the output weights, $v_i \in \mathbb{R}^n$ and $v_{i,0}$ $(i = 0, \ldots, k)$ are the input weights, and $\sigma : \mathbb{R} \to \mathbb{R}$, the *activation function*, is the *standard sigmoid function*, given by

$$\sigma(\alpha) = \frac{1}{1 + e^{-\alpha}}. \tag{1.1}$$

This function is illustrated in Figure 1.3. Each of the functions

$$x \mapsto \sigma\left(v_i \cdot x + v_{i,0}\right)$$

can be thought of as a smoothed version of the function computed by a simple perceptron. Thus, the two-layer sigmoid network computes an affine combination of these 'squashed' affine functions. It should be

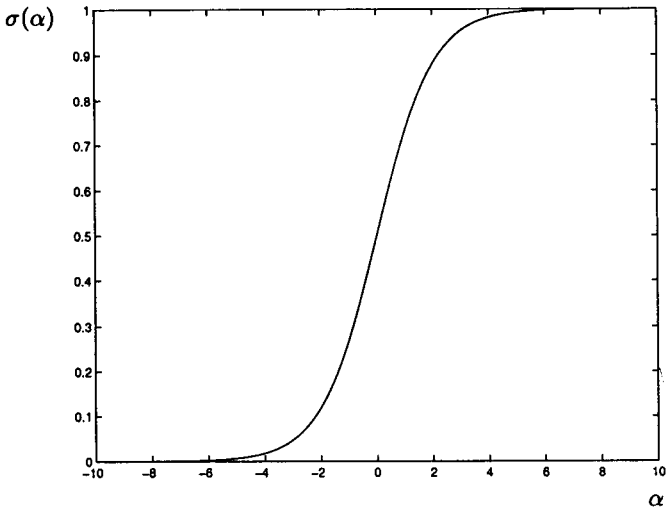

Fig. 1.3. The graph of the function $\sigma(\cdot)$ defined in Equation (1.1).

noted that the output of this network is a real number, and is not simply either 0 or 1 as for the simple perceptron. To use a network of this kind for a supervised learning problem, a learning algorithm would receive a set of labelled examples ((x, y) pairs, with $x \in \mathbb{R}^n$ and $y \in \mathbb{R}$) and attempt to find parameters that minimize some measure of the error of the network output over the training data. One popular technique is to start with small initial values for the parameters and use a 'gradient descent' procedure to adjust the parameters in such a way as to locally minimize the sum over the training examples (x_i, y_i) of the squared errors $(f(x_i) - y_i)^2$. In general, however, this approach leads only to a local minimum of the squared error.

We can consider the key issues of approximation, estimation, and computation for this network also. The approximation question has a more positive answer in this case. It is known that two-layer sigmoid networks are 'universal approximators', in the sense that, given any continuous function f defined on some compact subset S of $\mathbb{R}^n$, and any desired accuracy ϵ, there is a two-layer sigmoid network computing a function that is within ϵ of f at each point of S. Of course, even though such a network exists, a limited amount of training data might not provide enough information to specify it accurately. How much

data *will* suffice depends on the complexity of the function f (or more precisely on the complexity—number of computation units and size of parameters—of a network that accurately approximates f). Results in Part 3 address these questions, and Part 4 considers the computational complexity of finding a suitable network.

General neural networks

Quite generally, a neural network N may be regarded as a machine capable of taking on a number of 'states', each of which represents a function computable by the machine. These functions map from an input space X (the set of all possible patterns) to an output space Y. For neural networks, inputs are typically encoded as vectors of real numbers (so $X \subseteq \mathbb{R}^n$ for some n), and these real numbers often lie in a bounded range. In Part 1, we consider binary output networks for classification problems, so, there, we have $Y = \{0, 1\}$. In Parts 2 and 3 we consider networks with real outputs.

Formalizing mathematically, we may regard a neural network as being characterized by a set Ω of states, a set X of inputs, a set Y of outputs, and a parameterized function $F : \Omega \times X \to Y$. For any $\omega \in \Omega$, the function *represented by state* ω is $h_\omega : X \to Y$ given by

$$h_\omega(x) = F(\omega, x).$$

The function F describes the functionality of the network: when the network is in state ω it computes the function h_ω. The set of functions *computable by* N is $\{h_\omega : \omega \in \Omega\}$, and this is denoted by H_N. As a concrete example of this, consider the simple perceptron. Here, a typical state is $\omega = (w_1, w_2, \ldots, w_n, \theta)$, and the function it represents is

$$\begin{aligned} h_\omega(x) &= F(\omega, x) \\ &= F\left((w_1, w_2, \ldots, w_n, \theta), (x_1, x_2, \ldots, x_n)\right) \\ &= \operatorname{sgn}\left(\sum_{j=1}^{n} w_j x_j - \theta\right). \end{aligned}$$

1.3 Outline of the Book

The first three parts of the book define three supervised learning problems and study how the accuracy of a model depends on the amount

of training data and the model complexity. Results are generally of the form

$$\text{error} \leq (\text{estimate of error}) + (\text{complexity penalty}),$$

where the complexity penalty increases with some measure of the complexity of the class of models used by the learning system, and decreases as the amount of data increases. How 'complexity' is defined here depends both on the definition of error and on how the error is estimated. The three different learning problems are distinguished by the types of labels that must be predicted and by how the network outputs are interpreted.

In Part 1, we study the *binary classification problem*, in which we want to predict a binary-valued quantity using a class of binary-valued functions. The correct measure of complexity in this context is a combinatorial quantity known as the *Vapnik-Chervonenkis dimension*. Estimates of this dimension for simple perceptrons and networks of perceptrons have been known for some time. Part 1 reviews these results, and presents some more recent results, including estimates for the more commonly used sigmoid networks. In all cases, the complexity of a neural network is closely related to its size, as measured by the number of parameters in the network.

In Part 2, we study the *real classification problem*, in which we again want to predict a binary-valued quantity, but by using a class of real-valued functions. Learning algorithms that can be used for classes of real-valued functions are quite different from those used for binary-valued classes, and this leads to some anomalies between experimental experience and the VC theory described in Part 1. Part 2 presents some recent advances in the area of *large margin classifiers*, which are classifiers based on real-valued functions whose output is interpreted as a measure of the confidence in a classification. In this case, the correct measure of complexity is a scale-sensitive version of the VC-dimension known as the *fat-shattering dimension*. We shall see that this analysis can lead to more precise estimates of the misclassification probability (that is, better answers to the estimation question), and that the size of a neural network is not always the most appropriate measure of its complexity, particularly if the parameters are constrained to be small.

In Part 3, we study the *real prediction problem*. Here, the problem is to predict a real-valued quantity (using a class of real-valued functions). Once again, the fat-shattering dimension emerges as the correct measure of complexity. This part also features some recent results on the use of

convex function classes for real prediction problems. For instance, these results suggest that for a simple function class, using the convex hull of the class (that is, forming a two-layer neural network of functions from the class, with a constraint on the output weights) has considerable benefits and little cost, in terms of the rate at which the error decreases.

Part 4 concerns the *algorithmics* of supervised learning, considering the computational limitations on learning with neural networks and investigating the performance of particular learning algorithms (the perceptron algorithm and two constructive algorithms for two-layer networks).

1.4 Bibliographical Notes

There are many good introductory books on the topic of artificial neural networks; see, for example, (Hertz, Krogh and Palmer, 1991; Haykin, 1994; Bishop, 1995; Ripley, 1996; Anderson and Rosenfeld, 1988). There are also a number of books on the estimation questions associated with general learning systems, and many of these include a chapter on neural networks. See, for example, the books by Anthony and Biggs (1992), Kearns and Vazirani (1995), Natarajan (1991a), Vidyasagar (1997), and Vapnik (1982; 1995).

The notion of segmenting the analysis of learning systems into the key questions of approximation, estimation and computation is popular in learning theory research (see, for instance, (Barron, 1994)).

The simple perceptron and perceptron learning algorithm were first discussed by Rosenblatt (1958). The notion of adjusting the strengths of connections in biological neurons on the basis of correlations between inputs and outputs was earlier articulated by Hebb (1949) who, in trying to explain how a network of living brain cells could adapt to different stimuli, suggested that connections that were used frequently would gradually become stronger, while those that were not used would fade away. A classic work concerning the power (and limitations) of simple perceptrons is the book by Minsky and Papert (1969). Around the time of the publication of this book, interest in artificial neural networks waned, but was restored in the early 1980's, as computational resources became more abundant (and with the popularization of the observation that gradient computations in a multi-layer sigmoid network could share intermediate calculations). See, for example, (Rumelhart, Hinton and Williams, 1986a; Rumelhart, Hinton and Williams, 1986b). Since this

time, there have been many international conferences concentrating on neural networks research.

The 'universal approximation' property of neural networks has been proved under many different conditions and in many different ways; see (Cybenko, 1989; Hornik, Stinchcombe and White, 1990; Leshno, Lin, Pinkus and Schocken, 1993; Mhaskar, 1993).