1

Introduction

Richard Dybowski and Vanya Gant

In this introduction we outline the types of neural network featured in this book and how they relate to standard statistical methods. We also examine the issue of the so-called 'black-box' aspect of neural network and consider some possible future directions in the context of clinical medicine. Finally, we overview the remaining chapters.

A few evolutionary branches

The structure of the brain as a complex network of multiply connected cells (*neural networks*) was recognized in the late 19th century, primarily through the work of the Italian cytologist Golgi and the Spanish histologist Ramón y Cajal.[1] Within the reductionist approach to cognition (Churchland 1986), there appeared the question of how cognitive function could be modelled by artificial versions of these biological networks. This was the initial impetus for what has become a diverse collection of computational techniques known as *artificial neural networks* (ANNs).

The design of artificial neural networks was originally motivated by the phenomena of learning and recognition, and the desire to model these cognitive processes. But, starting in the mid-1980s, a more pragmatic stance has emerged, and ANNs are now regarded as non-standard statistical tools for pattern recognition. It must be emphasized that, in spite of their biological origins, they are not 'computers that think', nor do they perform 'brain-like' computations.

The 'evolution' of artificial neural networks is divergent and has resulted in a wide variety of 'phyla' and 'genera'. Rather than examine the development of every branch of the evolutionary tree, we focus on those associated with the types of ANN mentioned in this book, namely multilayer perceptrons (Chapters 2–8, 10–13), radial basis function networks (Chapter 12), Kohonen feature maps (Chapters 2, 5), adaptive resonance theory networks (Chapters 2, 9), and neuro-fuzzy networks (Chapters 10, 12).

We have not set out to provide a comprehensive tutorial on ANNs; instead, we

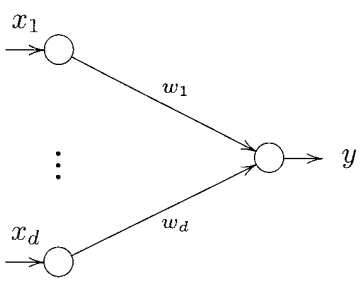

Figure 1.1. A graphical representation of a McCulloch–Pitts neuron, and also of a single-layer perceptron. In the former, a discontinuous step function is applied to the weighted sum $w_0 + w_1x_1 + \cdots + w_dx_d$ to produce the output y; in the latter, the step function is replaced by a continuous sigmoidal function.

have suggested sources of information throughout the text, and we have provided some recommended reading in Appendix 1.1.

Multilayer perceptrons

At the start of the 20th century, a number of general but non-mathematical theories of cognition existed, such as those of Helmholtz and Pavlov. At the University of Pittsburgh in the 1920s, Nicolas Rashevsky, a physicist, began a research programme to place biology within the framework of mathematical physics. This involved a number of projects, including an attempt to mathematically model Pavlovian conditioning in terms of neural networks (Rashevsky 1948). He continued his work at the University of Chicago, where he was joined by Warren McCulloch, a neuroanatomist, and then, in 1942, by a mathematical prodigy called Walter Pitts. Together, McCulloch & Pitts (1943) devised a simple model of the neuron. In this model (Figure 1.1), the input signals $x_1, \ldots, x_d$ to a neuron are regarded as a weighted sum $w_0 + w_1x_1 + \cdots + w_dx_d$. If the sum exceeds a predefined threshold value, the output signal y from the neuron equals 1; otherwise, it is 0. However, a McCulloch–Pitts neuron by itself is capable only of simple tasks, namely discrimination between sets of input values separable by a (possibly multidimensional) plane. Furthermore, the weights required for the neurons of a network had to be provided as no method for automatically determining the weights was available at that time.

Rosenblatt (1958) proposed that the McCulloch–Pitts neuron could be the basis of a system able to distinguish between patterns originating from different classes. The system, which he dubbed a *perceptron*, was a McCulloch–Pitts neuron with preprocessed inputs.[2] Motivated by Hebb's (1949) hypothesis that learning is based on the reinforcement of active neuronal connections, Rosenblatt (1960,

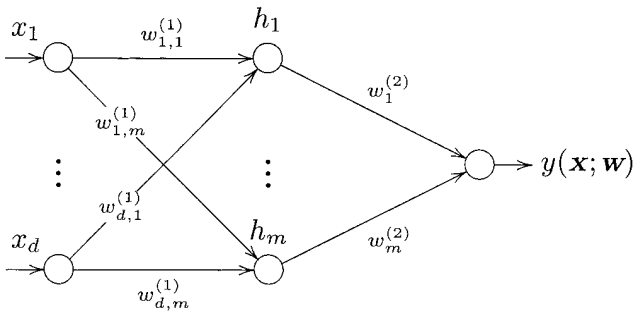

Figure 1.2. A multilayer perceptron with two layers of weights. The first layer of nodes, which receive the inputs $x_1,\ldots,x_d$, is called the *input layer.* The layer of nodes producing the output values is called the *output layer.* Layers of nodes between the input and output layers are referred to as *hidden layers.* The weighted sum h_j at the j-th hidden node is given by $w_{0,j}^{(1)} + w_{1,j}^{(1)} + \cdots w_{d,j}^{(1)} x_d$. The value from the j-th hidden node to the output node is a function f_{hid} of h_j, and the output $y(\boldsymbol{x};\boldsymbol{w})$ is a function of f_{out} of the weighted sum $w_0^{(2)} + w_1^{(2)} f_{\text{hid}}(h_1) + \cdots + w_m^{(2)} f_{\text{hid}}(h_m)$. Functions f_{hid} and f_{out} are typically sigmoidal. Note that a multilayer perceptron can have more than one layer of hidden nodes and more than one node providing output values.

1962) developed the *perceptron learning rule* and its associated convergence theorem. This solved the problem of a McCulloch–Pitts neuron 'learning' a set of weights. A number of workers (e.g. Block 1962) proved that the learning rule, when applied to a perceptron consisting of only a single layer of weights, would always modify the weights so as to give the optimal planar decision boundary possible for that perceptron.

Multilayer perceptrons (MLPs) are perceptrons having more than one layer of weights (Figure 1.2), which enables them to produce complex decision boundaries. Unfortunately, as pointed out by Minsky & Papert (1969), the perceptron learning rule did not apply to MLPs,[3] a fact that severely limited the types of problem to which perceptrons could be applied. This caused many researchers to leave the field, thereby starting the 'Dark Ages' of neural networks, during which little research was done. The turning point came in the mid-1980s when the back-propagation algorithm for training multilayer perceptrons was discovered independently by several researchers (LeCun 1985; Parker 1985; Rumelhart et al. 1986).[4] This answered the criticisms of Minsky & Papert (1969), and the Renaissance of neural networks began.

Multilayer perceptrons with sigmoidal hidden node functions are the most commonly used ANNs, as exemplified by the contributions to this book and the reviews by Baxt (1995) and Dybowski & Gant (1995). Each hidden node in Figure 1.2 produces a hyperplane boundary in the multidimensional space containing the input data. The output node smoothly interpolates between these boundaries to give decision regions of the input space occupied by each class of interest. With a

single logistic output unit, MLPs can be viewed as a non-linear extension of logistic regression, and, with two layers of weights, they can approximate any continuous function (Blum & Li 1991).[5] Although training an MLP by back-propagation can be a slow process, there are faster alternatives such as *Quickprop* (Fahlman 1988).

A particularly eloquent discussion of MLPs is given by Bishop (1995, Chap. 4) in his book *Neural Networks for Pattern Recognition.*

A statistical perspective on multilayer perceptrons

The genesis and renaissance of ANNs took place within various communities, and articles published during this period reflect the disciplines involved: biology and cognition, statistical physics, and computer science. But it was not until the early 1990s that a probability-theoretic perspective emerged, with Bridle (1991), Ripley (1993), Amari (1993) and Cheng & Titterington (1994) being amongst the first to regard ANNs as being within the framework of statistics. The statistical aspect of ANNs has also been highlighted in textbooks by Smith (1993), Bishop (1995) and Ripley (1996).

A recurring theme of this literature is that many ANNs are analogous to, or identical with, existing statistical techniques. For example, a popular statistical method for modelling the relationship between a binary response variable y and a vector (an ordered set) of covariates $\boldsymbol{x}$ is *logistic regression* (Hosmer & Lemeshow 1989; Collett 1991), but consider the single-layer perceptron of Figure 1.1:

$$y(\boldsymbol{x};\boldsymbol{w}) = f_{\text{out}}\left(w_0 + \sum_{i=1}^{d} w_i x_i\right). \tag{1.1}$$

If the output function f_{out} of Eq. (1.1) is logistic,

$$f_{\text{out}}(r) = 1 + \exp[-(r)]^{-1},$$

(where r is any value) and the perceptron is trained by a cross-entropy error function, Eq. (1.1) will be functionally identical with a main-effects logistic regression model

$$\hat{p}(y = 1 \mid \boldsymbol{x}) = \left\{1 + \exp\left[-(\hat{\beta}_0 + \sum_{i=1}^{d} \hat{\beta}_i x_i)\right]\right\}^{-1}.$$

Using the notation of Figure 1.2, the MLP can be written as

$$y(\boldsymbol{x};\boldsymbol{w}) = f_{\text{out}}\left(w_0^{(2)} + \sum_{j=1}^{m} w_j^{(2)} f_{\text{hid}}\left(w_{0,j}^{(1)} + \sum_{i=1}^{d} w_{i,j}^{(1)} x_i\right)\right), \tag{1.2}$$

but Hwang et al. (1994) have indicated that Eq. (1.2) can be regarded as a

particular type of projection pursuit regression model when f_{out} is linear:

$$y(\boldsymbol{x}; \boldsymbol{w}) = v_0 + \sum_{j=1}^{m} v_j f_j \left(u_{0,j} + \sum_{i=1}^{d} u_{i,j} x_i \right). \tag{1.3}$$

Projection pursuit regression (Friedman & Stuetzle 1981) is an established statistical technique and, in contrast to an MLP, each function f_j in Eq. (1.3) can be different, thereby providing more flexibility.[6] However, Ripley and Ripley (Chapter 11) point out that the statistical algorithms for fitting projection pursuit regression are not as effective as those for fitting MLPs.

Another parallel between neural and statistical models exists with regard to the problem of overfitting. In using an MLP, the aim is to have the MLP generalize from the data rather than have it fit to the data (*overfitting*). Overfitting can be controlled for by adding a *regularization function* to the error term (Poggio et al. 1985). This additional term penalizes an MLP that is too flexible. In statistical regression the same concept exists in the form of the *Akaike information criterion* (Akaike 1974). This is a linear combination of the deviance and the number of independent parameters, the latter penalizing the former. Furthermore, when regularization is implemented using weight decay (Hinton 1989), a common approach, the modelling process is analogous to ridge regression (Montgomery & Peck 1992, pp. 329–344) – a regression technique that can provide good generalization.

One may ask whether the apparent similarity between ANNs and existing statistical methods means that ANNs are redundant within pattern recognition. One answer to this is given by Ripley (1996, p. 4):

> The traditional methods of statistics and pattern recognition are either *parametric* based on a family of models with a small number of parameters, or *non-parametric* in which the models used are totally flexible. One of the impacts of neural network methods on pattern recognition has been to emphasize the need in large-scale practical problems for something in between, families of models with large but not unlimited flexibility given by a large number of parameters. The two most widely used neural network architectures, *multi-layer perceptrons* and *radial basis functions* (RBFs), provide two such families (and several others already in existence).

In other words, ANNs can act as *semi-parametric* classifiers, which are more flexible than parametric methods (such as the quadratic discriminant function (e.g. Krzanowski 1988)) but require fewer model parameters than non-parametric methods (such as those based on kernel density estimation (Silverman 1986)). However, setting up a semi-parametric classifier can be more computationally intensive than using a parametric or non-parametric approach.

Another response is to point out that the widespread fascination for ANNs has

attracted many researchers and potential users into the realm of pattern recognition. It is true that the neural-computing community rediscovered some statistical concepts already in existence (Ripley 1996), but this influx of participants has created new ideas and refined existing ones. These benefits include the *learning of sequences* by time delay and partial recurrence (Lang & Hinton 1988; Elman 1990) and the creation of powerful visualization techniques, such as *generative topographic mapping* (Bishop et al. 1997). Thus the ANN movement has resulted in statisticians having available to them a collection of techniques to add to their repertoire. Furthermore, the placement of ANNs within a statistical framework has provided a firmer theoretical foundation for neural computation, and it has led to new developments such as the Bayesian approach to ANNs (MacKay 1992).

Unfortunately, the rebirth of neural networks during the 1980s has been accompanied by hyperbole and misconceptions that have led to neural networks being trained incorrectly. In response to this, Tarassenko (1995) highlighted three areas where care is required in order to achieve reliable performance: firstly, there must be sufficient data to enable a network to generalize effectively; secondly, informative features must be extracted from the data for use as input to a network; thirdly, balanced training sets should be used for underrepresented classes (or *novelty detection* used when abnormalities are very rare (Tarassenko et al. 1995)). Tarassenko (1998) discussed these points in detail, and he stated:

> It is easy to be carried away and begin to overestimate their capabilities. The usual consequence of this is, hopefully, no more serious than an embarrassing failure with concomitant mutterings about black boxes and excessive hype. Neural networks cannot solve every problem. Traditional methods may be better. Nevertheless, neural networks, when they are used wisely, usually perform at least as well as the most appropriate traditional method and in some cases significantly better.

It should also be emphasized that, even with correct training, an ANN will not necessarily be the best choice for a classification task in terms of accuracy. This has been highlighted by Wyatt (1995), who wrote:

> Neural net advocates claim accuracy as the major advantage. However, when a large European research project, StatLog, examined the accuracy of five ANN and 19 traditional statistical or decision-tree methods for classifying 22 sets of data, including three medical datasets [Michie et al. 1994], a neural technique was the most accurate in only one dataset, on DNA sequences. For 15 (68%) of the 22 sets, traditional statistical methods were the most accurate, and those 15 included all three medical datasets.

But one should add the comment made by Michie et al. (1994, p. 221) on the results of the StatLog project:

> With care, neural networks perform very well as measured by error rate. They seem to provide either the best or near best predictive performance in nearly all cases . . .

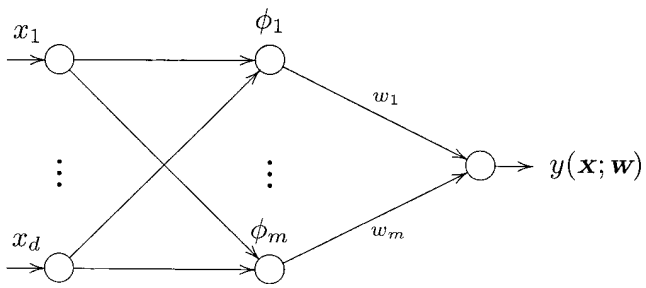

Figure 1.3. A radial basis function network. The network has a single layer of basis functions between the input and output layers. The value of ϕ_j produced by the *j*-th basis function is dependent on the distance between the 'centre' $\boldsymbol{x}^{[j]}$ of the function and the vector of input values $x_1, \ldots, x_d$. The output $y(\boldsymbol{x};\boldsymbol{w})$ is the weighted sum $w_0 + w_1\phi_1 + \cdots + w_m\phi_m$. Note that a radial basis function network can have more than one output node, and the functions $\phi_1, \ldots, \phi_m$ need not be the same.

Nevertheless, when an ANN is being evaluated, its performance must be compared with that obtained from one or more appropriate standard statistical techniques.

Radial basis function networks

Unlike MLPs, a number of so-called 'neural networks' were not biologically motivated, and one of these is the radial basis function network. Originally conceived in order to perform multivariate interpolation (Powell 1987), *radial basis function networks* (RBFNs) (Broomhead & Lowe 1988) are an alternative to MLPs. Like an MLP, an RBFN has input and output nodes; but there the similarity ends, for an RBFN has a middle layer of radially symmetric functions called *basis functions*, each of which can be designed separately (Figure 1.3). The idea of using basis functions originates from the concept of potential functions proposed by Bashkirov et al. (1964) and illustrated by Duda & Hart (1973).

Each basis function can be regarded as being centred on a prototypic vector of input values. When a vector of values is applied to an RBFN, a measure of the proximity of the vector to each of the prototypes is determined by the corresponding basis functions, and a weighted sum of these measures is given as the output of the RBFN (Figure 1.3).

The basis functions define *local responses* (*receptive fields*) (Figure 1.4). Typically, only some of the hidden units (basis functions) produce significant values for the final layers. This is why RBFNs are sometimes referred to as *localized receptive field networks*. In contrast, all the hidden units of an MLP are involved in determining the output from the network (they are said to form a *distributed representation*). The receptive field approach can be advantageous when the

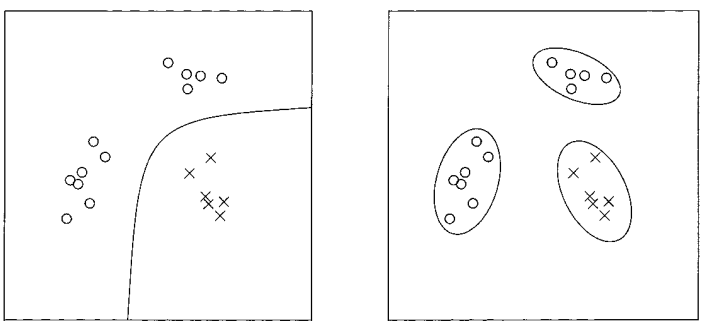

Figure 1.4. Schematic representation of possible decision regions created by (a) the hyperplanes of a multilayer perceptron, and (b) the kernel functions of a radial basis function network. The circles and crosses represent data points from two respective classes.

distribution of the data in the space of input values is multimodal (Wilkins et al. 1994). Furthermore, RBFNs can be trained more quickly than MLPs (Moody & Darken 1989), but the number of basis functions required can grow exponentially with the number of input nodes (Hartman et al. 1990), and an increase in the number of basis functions increases the time taken, and amount of data required, to train an RBFN adequately.

Under certain conditions (White 1989; Lowe & Webb 1991; Nabney 1999), an RBFN can act as a classifier. An advantage of the local nature of RBFNs compared with MLP classifiers is that a new set of input values that falls outside all the localized receptor fields could be flagged as not belonging to any of the classes represented. In other words, the set of input values is novel. This is a more cautious approach than the resolute classification that can occur with MLPs, in which a set of input values is always assigned to a class, irrespective of the values. For further details on RBFNs, see Bishop (1995, Chap. 5).

A statistical perspective on radial basis function networks

A simple linear discriminant function (Hand 1981, Chap. 4) has the form

$$g(\boldsymbol{x}) = w_0 + \sum_{i=1}^{d} w_i x_i, \tag{1.4}$$

with $\boldsymbol{x}$ assigned to a class of interest if $g(\boldsymbol{x})$ is greater than a predefined constant. This provides a planar decision surface and is functionally equivalent to the McCulloch–Pitts neuron. Equation (1.4) can be generalized to a linear function of functions, namely a *generalized linear discriminant function*

$$g(\boldsymbol{x}) = w_0 + \sum_{i=1}^{m} w_i f(\boldsymbol{x}), \tag{1.5}$$

which permits the construction of non-linear decision surfaces. If we represent an RBFN by the expression

$$g(\boldsymbol{x}) = w_0 + \sum_{i=1}^{m} w_i \phi_i(\| \boldsymbol{x} - \boldsymbol{x}^{[i]} \|), \tag{1.6}$$

where $\| \boldsymbol{x} - \boldsymbol{x}^{[i]} \|$ denotes the distance (usually Euclidean) between input vector $\boldsymbol{x}$ and the 'centre' $\boldsymbol{x}^{[i]}$ of the i-th basis function ϕ_i, comparison of Eq. (1.5) with Eq. (1.6) shows that an RBFN can be regarded as a type of generalized linear discriminant function.

Multilayer perceptrons and RBFNs are trained by *supervised learning*. This means that an ANN is presented with a set of examples, each example being a pair $(\boldsymbol{x}, t)$, where $\boldsymbol{x}$ is a vector of input values for the ANN, and t is the corresponding target value, for example a label denoting the class to which $\boldsymbol{x}$ belongs. The training algorithm adjusts the parameters of the ANN so as to minimize the discrepancy between the target values and the outputs produced by the network.

In contrast to MLPs and RBFNs,the ANNs in the next two sections are based on unsupervised learning. In *unsupervised learning*, there are no target values available, only input values, and the ANN attempts to categorize the inputs into classes. This is usually done by some form of clustering operation.

Kohonen feature maps

Many parts of the brain are organized in such a way that different sensory inputs are mapped to spatially localized regions within the brain. Furthermore, these regions are represented by *topologically ordered maps*. This means that the greater the similarity between two stimuli, the closer the location of their corresponding excitation regions. For example, visual, tactile and auditory stimuli are mapped onto different areas of the cerebral cortex in a topologically ordered manner (Hubel & Wiesel 1977; Kaas et al. 1983; Suga 1985). Kohonen (1982) was one of a group of people (others include Willshaw & von der Malsburg (1976)) who devised computational models of this phenomenon.

The aim of Kohonen's (1982) *self-organizing feature maps* (SOFMs) is to map an input vector to one of a set of neurons arranged in a lattice, and to do so in such a way that positions in input space are topologically ordered with locations on the lattice. This is done using a training set of input vectors $\boldsymbol{\xi}(1), \ldots, \boldsymbol{\xi}(m)$ and a set of prototype vectors $\boldsymbol{w}(1), \ldots, \boldsymbol{w}(n)$ in input space. Each prototype vector $\boldsymbol{w}(i)$ is associated with a location $S(i)$ on (typically) a lattice (Figure 1.5).

As the SOFM algorithm presents each input vector $\boldsymbol{\xi}$ to the set of prototype vectors, the vector $\boldsymbol{w}(i^*)$ nearest to $\boldsymbol{\xi}$ is moved towards $\boldsymbol{\xi}$ according to a learning

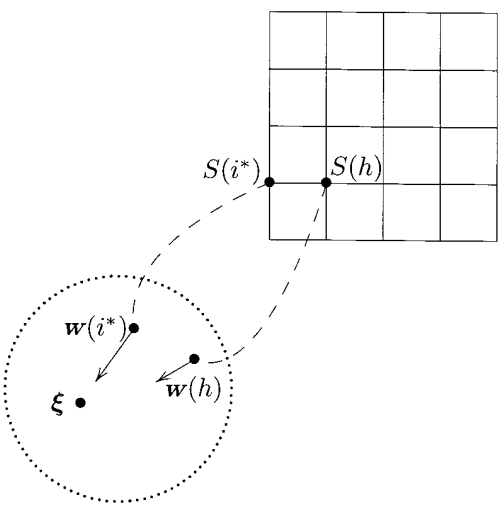

Figure 1.5. A graphical depiction of Kohonen's self-organizing feature map. See pp. 9–10 for an explanation. The lattice is two-dimensional, whereas data point (input vector) ξ and prototype vectors $\boldsymbol{w}(i^*)$ and $\boldsymbol{w}(h)$ reside in a higher-dimensional (input) space.

rule. In doing so, the algorithm also 'drags' towards $\boldsymbol{\xi}$ (but to a lesser extent) those prototype vectors whose associated locations on the lattice are closest to $S(i^*)$, where $S(i^*)$ is the lattice location associated with $\boldsymbol{w}(i^*)$. For example, $\boldsymbol{w}(h)$ in Figure 1.5 is dragged along with $\boldsymbol{w}(i^*)$ towards $\boldsymbol{\xi}$. Hertz et al. (1991) likened this process to an elastic net, existing in input space, which wants to come as close as possible to $\boldsymbol{\xi}(1), \ldots, \boldsymbol{\xi}(m)$. The coordinates of the intersections of the elastic net are defined by the prototype vectors $\boldsymbol{w}(1), \ldots, \boldsymbol{w}(n)$. If successful, two locations $S(j)$ and $S(k)$ on the lattice will be closer to each other the closer their associated prototype vectors $\boldsymbol{w}(j)$ and $\boldsymbol{w}(k)$ are positioned in input space.

The SOFM algorithm provides a means of visualizing the distribution of data points in input space, but, as pointed out by Bishop (1995), this can be weak if the data do not lie within a two-dimensional subspace of the higher-dimensional space containing the data. Another problem with SOFM is that the 'elastic net' could twist as it moves towards the training set, resulting in a distorted visualization of the data (e.g. Hagan et al. 1996).

For those wishing to know more about SOFMs, we recommend the book *Neural Computation and Self-Organizing Maps* by Ritter et al. (1992).

Adaptive resonance theory networks

A feature of cognitive systems is that they can be receptive to new patterns (described as *plasticity*) but remain unchanged to others (described as *stability*).