1 Introduction

Artificial neural networks are nonlinear mapping systems whose structure is loosely based on principles observed in the nervous systems of humans and animals. The major parts of a real neuron include (see figure 1.1) a branching dendritic tree that contacts and collects signals from other neurons, a cell body that integrates the signals and generates a response, and a branching axon that distributes the response to other neurons. The response of each neuron is a nonlinear function of its inputs and internal state. It is thought to be largely determined by the input connection strengths. Real neurons are more complex than this, of course, but still simple in comparison to the entire brain. The interesting idea is that massive systems of simple units linked together in appropriate ways can generate many complex and interesting behaviors.

Artificial neural networks are loosely based on these ideas. In general terms, an artificial neural network consists of a large number of simple processors linked by weighted connections. By analogy, the processing units may be called neurons. Each unit receives inputs from many other nodes and generates a single scalar output that depends only on locally available information, either stored internally or arriving via the weighted connections. The output is distributed to and acts as an input to other processing nodes.

By itself, a single processing element is not very powerful; the power of the system emerges from the combination of many units in a network. A network is specialized to implement different functions by varying the connection topology and values of the connecting weights. Depending on the connections, many complex functions can be realized. In fact, it has been shown that a sufficiently large network with an appropriate structure and properly chosen weights can approximate with arbitrary accuracy any function satisfying certain broad constraints.

In many networks, the processing units have responses like (see figure 1.1b)

$$y = f\left(\sum_k w_k x_k\right) \tag{1.1}$$

where x_k are the output signals of other nodes or external system inputs, w_k are the weights of the connecting links, and $f(\cdot)$ is a simple nonlinear function. Here, the unit computes a weighted linear combination of its inputs and passes this through the nonlinearity f to produce a scalar output. In general, f is a bounded nondecreasing nonlinear function such as the *sigmoid* function (figure 1.2)

$$f(\mu) = \frac{1}{1+e^{-\mu}}. \qquad \text{(the sigmoid)}$$

The tanh and step functions are other common choices. f is sometimes called the *squashing function* because it limits very large positive or negative values. The term *perceptron*

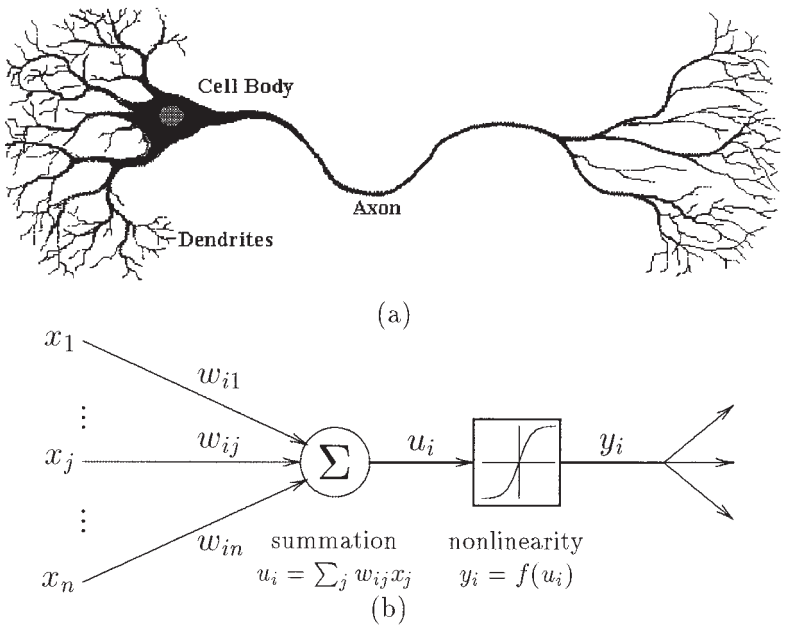

Figure 1.1
(a) A real neuron collects input signals from other neurons through a dendritic tree, integrates the information, and distributes its response to other neurons via the axon. (b) An artificial neuron model.

is commonly used to refer to any feedforward network of nodes with responses like equation 1.1.

A network can have arbitrary structure, but layered architectures are very popular. The multilayer perceptron (figure 1.3) is widely used. In this structure, units are arranged in layers and connected so that units in layer L receive inputs from the preceding layer $L-1$ and send outputs to the following layer $L+1$. External inputs are applied at the first layer and system outputs are taken at the last layer. Internal layers not observable from the inputs or outputs are called *hidden* layers. The simplest networks have just one active layer, the output units. (By convention, inputs are not counted as an active layer since they do no processing.) Single-layer networks are less powerful than multilayer circuits so applications are relatively limited.

The network in figure 1.3 has a *feedforward* structure, meaning there are no connection loops that would allow outputs to feed back to their inputs (perhaps indirectly) and change the output at a later time. The network implements a static mapping that depends only on its present inputs and is independent of previous system states. Although recurrent networks

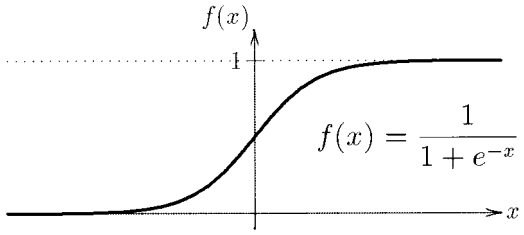

Figure 1.2
The sigmoid is a continuous, bounded, monotonic function of its input x. It saturates at 0 for large negative inputs and at 1 for large positive inputs. Near zero, it is approximately linear.

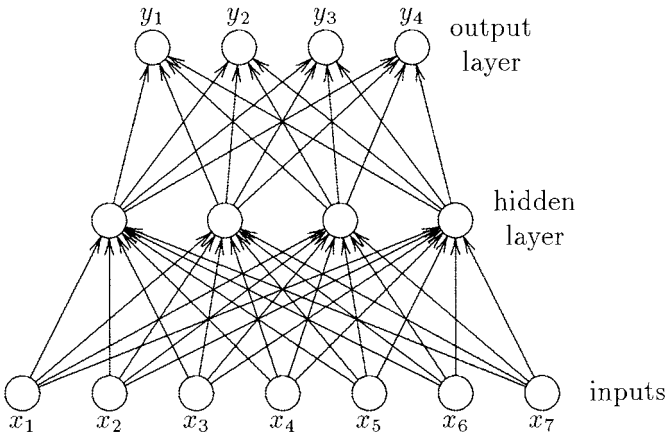

Figure 1.3
A layered feedforward network.

with feedback have a wider range of possible behaviors, analysis and training are more difficult. This book focuses on feedforward models, which form an important subclass in themselves.

The network in figure 1.3 is also *fully connected*. That is, every node in layer L receives inputs from every node in the preceding layer $L-1$ and projects outputs to every node in the following layer $L+1$.

This model is, of course, a drastically simplified approximation of real nervous systems. This isn't the place to dwell on its limitations; the model captures certain major characteristics which are thought to be important and undoubtedly ignores others. A few differences between multilayer perceptrons and real nervous systems are listed in table 1.1. More

Table 1.1
Artificial Neural Networks versus Real Nervous Systems.

MLP ANN	Real nervous system
feedforward	recurrent
fully-connected	mostly local connections
uniform structure	functional modules
a few node types	hundreds of neuron types
10–1000 nodes	human brain: $O(10^{11})$ neurons, $O(10^{15})$ synapses
mostly static	dynamic: spike trains, facilitation, fatigue, synchronization

details can be found in Kandel and Schwartz [206], a comprehensive introductory reference on the structure of the human nervous system.

In spite of its limitations, the model is rich enough to have useful computational properties. It can also be argued that its simplicity is a virtue because it allows theoretical analysis, which may lead to further understanding and better models.

Like real nervous systems, one of the reasons why artificial neural networks are interesting is that they "learn from examples." That is, given examples of input patterns and desired outputs, algorithms exist for changing the weights so that the network produces the correct output for each training input. Back-propagation, discussed in chapter 5, is one of the most popular methods. There are practical difficulties, of course, so training is not always successful, but in principle a network can always be found to fit a well-behaved data set. If things go well, the system learns not only the training examples but also the underlying relationship that allows it to produce the correct output (or at least a reasonable output) in response to new inputs. Useful learning may occur in spite of noisy and sometimes misleading data, incomplete patterns, and other defects in the training set.

Because it is adaptive, the system is potentially self-repairing. In physical implementations, small manufacturing defects may be tolerated because the weights can be adjusted to 'train around' defective elements. Periodic retraining might also compensate for minor damage or parameter drift due to wear and aging. This is attractive because it could lead to lower manufacturing costs.

The system also derives some fault tolerance from its redundant parallel structure. Many inputs contribute to the output of each node and many nodes interact to produce the overall output so the system is relatively insensitive to minor damage. Functions are spread over many elements rather than isolated in a single location, so loss of a few elements degrades overall performance somewhat but generally does not cause a complete breakdown. This property has been called *graceful degradation*.

Another potential benefit of the parallel structure is very fast computation. A unit's output at time t depends only on its inputs at time $t - 1$ (and maybe some stored parameters).

Units in a layer are independent so they can all be evaluated simultaneously. The layers have a serial dependency so a feedforward network with L layers would respond to a change in the input after L time steps. A network with 2000 nodes in two layers, say, could produce an output in just two time steps instead of the 2000 steps that would be required if each node were evaluated serially.

At the present time, most networks are simulated on serial computers so not all of the potential advantages are realized. Even so, artificial neural networks are still very useful because of their functional mapping properties and the ability to learn from examples. Multilayer networks have been shown to have a certain "universal approximation" property. As noted, a sufficiently large network with the appropriate structure can, theoretically, approximate almost any desired function. Networks have been compared with many other functional approximation systems and found competitive in terms of accuracy. (Learning speed is sometimes a problem, however.) This and the ability to learn from examples allow modeling of complex systems whose behavioral rules are not known in detail or are difficult to analyze correctly. This includes the common case where a good theoretical model exists but depends on physical parameters that are difficult or expensive to measure. Networks offer the possibility of modeling the actual system behavior based on its observed input-output response instead of using a simplified and necessarily imperfect parametric model. By learning from real data, an adaptive system may be able to include nonlinear effects (e.g., friction and backlash in mechanical systems) that might be ignored in simplified theoretical models.

This book focuses on practical aspects of designing and training feedforward multilayer networks. It should be noted, however, that aside from possible technological applications, artificial neural networks are important as simplified models of real nervous systems and have been used to test theories about brain function, cognition, and learning. Indeed, much of the original work has been done, and continues, in fields such as psychology, neuroscience, and cognitive science. Even among many of those interested only in technological applications, the biological connection is part of the model's appeal since there is undoubtedly more that can be learned which may lead to more powerful models and better applications.

The preceding lists some of the promise of artificial neural networks. Unfortunately, difficulties do arise and learning is not always successful. This book focuses on practical aspects of multilayered perceptron networks—how to develop and use them in practical applications.

Chapter 2 describes the general idea of supervised learning, the process of training a system to perform a desired function. Chapter 3 describes the simplest possible systems,

single-layer networks, and properties that result from this structure. Chapter 4 considers multilayered networks and additional properties that result from the layered structure.

Chapter 5 introduces the back-propagation algorithm, the most popular method for training layered networks. The next several chapters consider how training parameters affect learning and describe some variations of the basic algorithm. Chapter 6 discusses effects of the learning rate and momentum parameters on learning time. Chapter 7 considers various methods for initializing the network structure and weights prior to training. Chapter 8 describes some properties of the error surface and how they affect the training process. Chapter 9 describes some variations of the basic back-propagation algorithm. Chapter 10 describes classical optimization methods that may perform better than back-propagation in some cases. Chapter 11 describes the genetic algorithm.

Chapters 12 and 13 consider methods for adapting the structure of the network to the problem. Chapter 12 considers constructive training methods, which start with a small network and add elements as required. Chapter 13 considers pruning methods, which take the opposite strategy and start with a large network and then remove elements as needed.

Learning from examples is a sampling process. Once a network learns the training data, it is reasonable to ask how well it generalizes to patterns outside the training set. Chapter 14 discusses factors influencing generalization. Chapter 15 continues by examining some theoretical methods for assessing and predicting generalization performance. Chapter 16 describes a number of heuristics for obtaining better generalization. Chapter 17 focuses on one particular heuristic and considers the effects of training with noisy input data.