CHAPTER 1

Overview: Artificial Neural Networks and Neural Computing

Here is Edward Bear, coming downstairs now, bump, bump, bump, on the back of his head, behind Christopher Robin. It is, as far as he knows, the only way of coming downstairs, but sometimes he feels that there really is another way, if only he could stop bumping for a moment and think of it.

Winnie-the-Pooh, ©1926 A. A. Milne

1.1 WHAT IS NEURAL COMPUTING?

1.1.1 Computing Architectures

The notion of computing takes many forms. Historically,[1] computing has been dominated by the concept of *programmed computing*, in which (usually procedural) algorithms are designed and subsequently implemented using the currently dominant architecture. An alternative viewpoint is needed when one considers the "computing" necessary in biological systems. For example, computation in the human brain is much different from the aforementioned paradigm in that

- The computation is massively distributed and parallel.
- Learning replaces a priori program development.

Taking these cues from nature, the new and biologically motivated computing paradigm of *artificial neural networks* (ANNs) has arisen. ANN technology has the potential to be a dominant computing architecture, and artificial neurons may become the "ultimate RISC (reduced-instruction-set computer) building block." This book explores the ramifications of designing a trained computation on a massively parallel system, including current ideas and approaches, mathematical underpinnings, and application examples.

Emerging ANN technology is a broad body of often loosely related knowledge and techniques that provides practical alternatives to "conventional" computing solutions and offers some potential for approaching many currently unsolved problems. Since no single technology is always the optimal solution for a given problem, the system designer must carefully trade off conventional solutions against the ANN alternative.

[1] That is, over the last 50 years.

ANNs represent both a field of science and a technology, where science is loosely defined as structured knowledge (usually concerned with the physical world) and technology represents applied science. In this book we explore both aspects, from the biological and mathematical underpinnings of ANNs to the engineering of ANN-based problem solutions. Recent advances in computer hardware, especially the realization of (relatively) inexpensive, massively parallel systems, have made ANN implementation more practical.

1.1.2 Chapter Overview

Three fundamental questions that should be addressed in any initial look at ANN technology are

1. Is ANN technology really new, and how can it be used?
2. What is common to or derived from other technologies?
3. What is common to all the ANNs we will study?

In this chapter, an overview of ANN technology, especially as it relates to other technologies, is presented. A brief history of the field is given. The major common aspects of neural computing, such as network topology, unit characteristics, black-box behavior, and training, are introduced to provide both perspective and a foundation for the detailed explorations of subsequent chapters.

1.1.3 Definition of Artificial Neural Network

What follows is a working and somewhat generic definition[2] of the device we will study. Throughout the remainder of the book, this definition will be refined and specialized.

> **ARTIFICIAL NEURAL NETWORK.** A structure (network) composed of a number of interconnected units (artificial neurons). Each unit has an input/output (I/O) characteristic and implements a local computation or function. The output of any unit is determined by its I/O characteristic, its interconnection to other units, and (possibly) external inputs. Although "hand crafting" of the network is possible, the network usually develops an overall functionality through one or more forms of training.

ANNs do not constitute one network, but a diverse *family* of networks. The overall function or functionality achieved is determined by the network topology, the individual neuron characteristics, and the learning or training strategy and training data.

To be useful, an ANN must have a means of interfacing with the outside world. Although not required by the preceding definition, typically the unit input/output (I/O) characteristics are simple (and common to all units), and the number of units is quite large. Note that the definition forces us to distinguish between a single unit (see Chapter 3) and a network. Finally, the computational structures we develop here may be implemented in a number of nonbiological ways, most typically through electronic elements. Therefore, the descriptor "artificial" is often assumed.

---

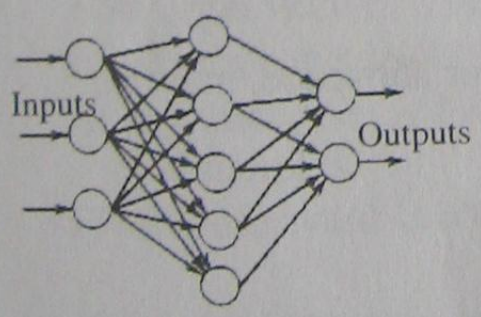
[2]Readers should note that numerous alternative definitions exist.

1.1.4 Fundamental Neural Network Concepts

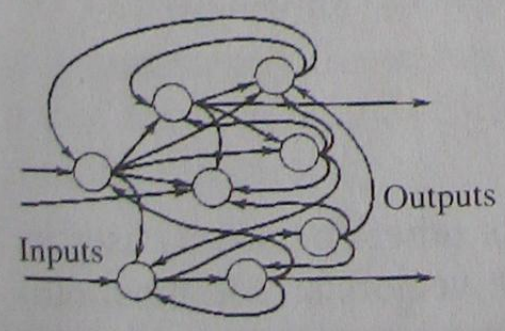
The following are key aspects of neural computing:

- As the definition of Section 1.1.3 indicates, the overall computational model consists of a *reconfigurable interconnection of simple elements, or units.* Figure 1.1 depicts two small-scale sample networks, where units are denoted by circles and interconnections are shown as arcs. Figure 1.1(*a*) depicts a *nonrecurrent* interconnection strategy, containing no closed interconnection paths. Note the depiction of units grouped in layers. By contrast, Figure 1.1(*b*) illustrates a *recurrent* network interconnection strategy, where the arbitrary interconnection flexibility allows closed-loop (feedback) paths to exist. This allows the network to exhibit far more complex temporal dynamics compared with the (open-loop) strategy of Figure 1.1(*a*). Also note that network topologies may be either static or dynamic. Finally, notice that some units in Figure 1.1 interface directly with the outside world, whereas others are "hidden" or internal.

  These network connection structures are explored in more detail in Chapters 4–10. Note that graphical representations, with units depicted as nodes and directionally sensitive interconnections shown as arcs, are useful mechanisms to convey topology.
- Individual units implement a local function, and the overall network of interconnected units displays a corresponding functionality. Analysis of this functionality, except through training and test examples, is often difficult. Moreover, the application usually determines, via *specifications*, the required functionality; it is the role of the ANN designer to determine network parameters that satisfy these specifications. In Chapter 3 we concentrate on characteristics of the individual units.
- Modifying patterns of interelement connectivity as a function of training data is a key learning approach. In other words, the system knowledge, experience, or training is stored in the form of network interconnections.
- To be useful, neural systems must be capable of storing information (i.e., they must be "trainable"). Neural systems are trained in the hope that they will subsequently display correct associative behavior when presented with new patterns to recognize



(*a*)



(*b*)

FIGURE 1.1
Basic topologies for (*a*) nonrecurrent and (*b*) recurrent ANNs.

or classify. That is, the objective in the training process is for the network to develop an internal structure enabling it to correctly identify or classify new, similar patterns. We consider both supervised and unsupervised training paradigms.
- A neural network is a dynamic system; its state (e.g., unit outputs and interconnection strengths) changes over time in response to external inputs or an initial (unstable) state.

Succeeding chapters present more detailed discussion of these ANN characteristics.

1.1.5 Introductory Terminology and Notational Conventions

Key terms

We begin with a short glossary of salient concepts:

adaptive system: a system capable of adjusting its performance (usually parametrically) for increased demands or to accommodate uncertain operational environments
algorithm: a method or procedure for attaining a goal or solution
architecture: the organization of hardware or software
classification: the ability to assign an input observation to a category
crossover: a process used in genetic algorithms to emulate sexual reproduction
feature: something that characterizes a property of an object or situation
fuzzy logic: an extension to crisp logic in which truth values are not restricted to be binary
generalization: the ability to cover more examples; the opposite of specialization; the behavior of the network using inputs not in the training set, H
heuristic: a rule of thumb used but not guaranteed to solve a problem
inversion: determining the input from the given output and the system model
learning: see Section 1.7
network: an amalgamation of interconnected entities
search: a ubiquitous problem in which a *search space*, or subspace, must be explored and evaluated
topology: the structure of a network
training: similar to learning
training set: denoted H; see Section 1.7
unit: the "atomic" element of an ANN; implements a local mapping
VLSI: very large scale integration (of fabricated silicon devices), usually to increase processing or memory capabilities

Notation

Generally, each ANN publication (be it a book, paper, or other document) uses its own nomenclature. In addition, some authors like to denote vectors as columns; others use rows. To simplify the situation, the following notation, corresponding to major

concepts, is used throughout this book. A vector is denoted by underlining; all vectors are assumed to be column vectors.

$\underline{x}$: A general vector.
$\underline{i}$: A vector of inputs to either a unit or a net. The jth input is denoted i_j. The dimension of $\underline{i}$ is always d, although the addition of a bias input requires either redefining $\underline{i}$ or augmenting d by 1.
$\underline{o}$: A vector of unit outputs. Note that sometimes $\underline{o} = \underline{i}$, in the case of recurrent or layered nets. The dimension of $\underline{o}$ is always c.
$\underline{o}^s$: A vector indicating a desired stored state in a recurrent network.
o_i: Unit i output. Often $o_i = f(net_i)$.
$\underline{net}_i$ or net_i: The net activation vector or the net activation to the ith unit, respectively.
w_j: The weight corresponding to the jth input of a unit.
w_{ij}: The weight corresponding to the jth input of unit i. Alternatively, when units are interconnected, w_{ij} represents the interconnection strength from unit j to unit i. In training, $w_{ij}(k)$ is often used to denote a weight at time or iteration k.
$\underline{w}$ or $\underline{w}^p$: A vector of unit weights, corresponding to unit p. Often denoted $\underline{w}^T$ in forming net activation.
W: A matrix of weights. Element w_{ij}, at row i and column j, is defined above.
P: A stimulus matrix (also denoted as S) used to represent the input vectors to the network, usually in H. Often written

$$P = \begin{pmatrix} \underline{i}_1^T \\ \underline{i}_2^T \\ \underline{i}_3^T \\ \vdots \\ \underline{i}_n^T \end{pmatrix}$$

R: A response matrix, with structure similar to P.
$\underline{t}$: A target vector (Chapters 4–7).
E: An error measure (Chapters 6–7) or energy (Chapter 8).[3]
H: A training set, either labeled or unlabeled. H has cardinality n; i.e., it consists of n samples.
FF: Feedforward (net).

Three important variables

As noted in the preceding notational definitions, three variables are used frequently and consistently throughout the text: d is the number of network inputs, c is the number of network outputs (note that $d = c$ is possible), and n is the number of available training samples.

[3]Unfortunately, it is used for both.

1.2 NEURAL COMPUTING APPLICATIONS

1.2.1 Characteristics of Problems Suitable for ANNs

Emulation of biological system computational structures may yield superior computational paradigms for certain classes of problems. Among these are the class of NP-hard problems, which includes labeling problems, scheduling problems, search problems, and other constraint satisfaction problems; the class of pattern/object recognition problems, notably in vision and speech understanding; and the class of problems dealing with flawed, missing, contradicting, fuzzy, or probabilistic data. These problems are characterized by some or all of the following: a high-dimensional problem space; complex, unknown, or mathematically intractable interactions between problem variables; and a solution space that may be empty, contain a unique solution, or (most typically) contain a number of (almost equally) useful solutions. Furthermore (as the following list indicates), ANNs seem to offer solutions to problems that involve human sensory input, such as speech, vision, and handwriting recognition. *Note that what is not simple is the mapping of an arbitrary problem to a neural network solution.*

1.2.2 Sample ANN Applications

A comprehensive look at all applications for ANNs (attempted, successful, or envisioned) is impractical. However, a look at the popular press, journals (see Section 1.10), and conference proceedings provides illustrative examples. Applications include

Image processing and computer vision, including image matching, preprocessing, segmentation and analysis, computer vision (e.g., circuit board inspection), image compression, stereo vision, and processing and understanding of time-varying images.

Signal processing, including seismic signal analysis and morphology.

Pattern recognition, including feature extraction [Sau89], radar signal classification and analysis, speech recognition and understanding, fingerprint identification, character (letter or number) recognition, and handwriting analysis ("notepad" computers).

Medicine [PvG90], including electrocardiographic signal analysis and understanding, diagnosis of various diseases, and medical image processing.

Military systems, including undersea mine detection, radar clutter classification, and tactical speaker recognition.

Financial systems, including stock market analysis [RZF94], real estate appraisal, credit card authorization [Ott94], and securities trading [BvdBW94].

Planning, control, and search, including parallel implementation of constraint satisfaction problems (CSPs), solutions to Traveling Salesman–like CSPs, and control and robotics.

Artificial intelligence, including abductive systems and implementation of expert systems [Gal93].

Power systems, including system state estimation, transient detection and classification, fault detection and recovery, load forecasting, and security assessment.
Human factors (interfacing).

1.3 A BRIEF OVERVIEW OF NEURAL COMPUTING

1.3.1 Background

Biological system functionality

As discussed in Chapter 3, biological system functionality is based on interconnections of specialized physical cells called neurons. The adaptability, context-sensitive nature, error tolerance, large memory capacity, and real-time capability of biological information-processing systems (most notably the brain) suggests an alternative architecture to emulate. The mere fact that the basic computing element of the human information-processing system is relatively slow (in the millisecond range and thus ridiculously slow vis-à-vis electronic devices) yet the overall processing operation is achieved in a few hundred milliseconds suggests that *the basis of biological computation is a small number of serial steps, each occurring on a massively parallel scale.* Furthermore, in this inherently parallel architecture, each of the processing elements is locally connected and relatively simple.

Neuromorphic computing

The term *neuromorphic engineering*[4] refers to a new discipline based on the design and fabrication of artificial neural systems, such as vision systems, head-eye systems, and roving robots, whose architecture and design principles are based on those of biological nervous systems. Neuromorphic engineering has a wide range of applications, from nonlinear adaptive control of complex systems to the design of smart sensors. Many of the fundamental principles in this field, such as the use of learning methods and the design of parallel hardware, are inspired by biological systems.

1.3.2 What Are the Relevant Computational Properties of the Human Brain?

One of the paradoxical aspects of the human "mind" is that complete knowledge of the neural (physiological) architecture is available, yet characterization of the fundamental high-level computation remains a mystery. By analogy, imagine connecting a digital logic analyzer to a functioning computer CPU with a completely known and well-documented architecture. The signals to and from the CPU are all available in the form of time-varying traces of electrical waveforms. Although this system is completely

[4]Coined by Carver Mead of Caltech.

characterized in terms of micro-operations, analysis of these low-level waveforms and subsequent determination of the overall high-level computation being performed (for example, matrix inversion) is essentially impossible. This is analogous to the problem of inferring algorithms for intelligence by studying biological (neural) signals.

The number of individual processing units and the interconnection complexity of the human brain are enormous. Note, however, that *the primary purpose, application, and objective of the human brain is survival*. The time-evolved performance of human intelligence reflects (arguably) an attempt to optimize this objective. This distinguishing characteristic does not, however, reduce our interest in biological computation, since many of the supporting or ancillary functions of the biological system are those for which we desire neural emulation. In addition,

1. *The brain integrates and stores experiences*, which could be previous classifications or associations of input data. In this sense, it self-organizes experience.
2. *The brain considers new experiences in the context of stored experiences.* This often requires a type of high-level, symbolic, or iconic matching ability, as well as a means to "index" stored past experiences from different viewpoints. This suggests a context-addressable structure.
3. *The brain is able to make accurate predictions about new situations on the basis of previously self-organized experiences.* This suggests a generalization capability.
4. *The brain does not require perfect information.* It is tolerant of deformations of input patterns or perturbations in input data, including incompleteness. This suggests another form of generalization.
5. *The brain represents a fault-tolerant architecture*, in the sense that loss of a few neurons may be recoverable by adaptation of those remaining and perhaps additional training.
6. *The brain seems to have available, perhaps unused, neurons ready for use.* This suggests that the system does not stop learning once it is fielded.
7. *The brain does not provide, through microscopic or macroscopic examination of its activity, much useful information concerning its operation at a high level.* For example, a time-varying CAT scan of the human brain in the act of solving a math problem does not reveal anything useful about the solution procedure. From an engineering analogy, this is equivalent to the problem of instrumenting a computer chip with appropriate logic probes and, on the basis of these signals alone, inferring the high-level computation (e.g., matrix inversion, solution to a differential equation) being performed. Thus, the opacity of brain operation is often reflected in ANNs, which may provide solutions to problems but not *explainable* solutions. As we show later, it is sometimes possible to attach limited semantic interpretation of unit behavior in selected cases.
8. *The brain tends to cause behavior that is homeostatic*, meaning "in a state of equilibrium (stable) or tending toward such a state." This is an interesting feature found, at a much lower scale, in certain recurrent networks (e.g., the networks of Hopfield and Grossberg, covered in Chapters 8 and 9).

Note that the brain, although remarkable, can no longer compete in many tasks for which digital computers are currently available. For example, the brain is ill equipped to solve for roots of high-order polynomials, invert large-dimension matrices, and solve complex sets of differential equations.

1.3.3 Neural Approaches to Computation

Training versus programming

For a computer to be useful, an outside (user) interface is necessary. There are several ways for humans to communicate with computers. Traditionally, *programming* has been the dominant vehicle. However, programming involves a formal syntax and a spectrum of possible (application-sensitive) languages and requires considerable skilled personnel. An alternative typified by biological systems is *training*. For example, young children are not "programmed" but learn by example and adaptation [Sch90]. Of course, for the training approach to be feasible, the computer must be trainable and training data must be available. In this context, there are several viewpoints we may adopt in studying ANNs:

- Neural systems (ideally) behave as trainable, adaptive, and even self-organizing information systems.
- Neural networks develop a functionality based on training/sample data.
- Neural networks may provide computational architectures through training rather than explicit "design."

Mathematical models and simulation

Most present non–biologically oriented neural network research concerns the development, characterization, and extension of *mathematical neural network models*. A mathematical neural network model refers to a set of (usually) nonlinear, n-dimensional equations that characterize the overall network operation, as well as structure, unit (and network) dynamics, and training. Commonly, difference or differential equations are employed [Jef90].

It should be noted that much of what is known about the performance and success of ANNs has been discovered through the *simulation* of ANNs on digital computers. This simulation often requires tremendous computational resources and may require a modification of the actual ANN computational structure. Assessment of the true behavior of ANNs in actual applications will become possible as the massively parallel hardware that is necessary (see Chapter 12) becomes available.

"Connectionist" models and computing

The connectionist philosophy toward computing is based on the notion that many human computational processes are naturally carried out in a highly parallel fashion (which could, provided architectures exist, be emulated in machine intelligence) with significant interactions between processes. These processes are an alternative to the formal manipulation of symbolic expressions [Hin89]. In essence, the overall computation is distributed over a large number of (usually simple) computational units, where each unit provides a portion of the computational effort. One interpretation is that the burden of the computational process must fall on the connection structure of the network [Fel85]. Connectionist computing models may be shown to have a number of abstract properties relevant to their application in problems involving cognitive processes [AA82]. More specifically, however, the number of computational units is large, their connectivity is severely restricted (usually to be very local), and their internal complexity is limited. Thus, neural nets, as described

previously, satisfy this requirement (as do other computational structures, e.g., systolic arrays).

The connectionist approach is, in some ways, a generalization of the neural network concept, where the individual unit or "extended neuron" is allowed to be more complex than the neuron defined earlier.

1.3.4 Advantages and Disadvantages of ANNs

Because ANNs are a relatively new computational paradigm, it is probably safe to say that the advantages, disadvantages, applications, and relationships to traditional computing are not fully understood. Expectations (some might say "hype") for this area are high. Neural networks are particularly well suited for certain applications, especially *trainable pattern association.* The notion that artificial neural networks can solve all problems in automated reasoning, or even all mapping problems, is probably unrealistic.

Advantages

- Inherently massively parallel
- May be fault tolerant because of parallelism
- May be designed to be adaptive
- Little need for extensive characterization of problem (other than through the training set)

Disadvantages

- No clear rules or design guidelines for arbitrary application
- No general way to assess the internal operation of the network
- Training may be difficult or impossible
- Difficult to predict future network performance (generalization)

1.4 ENGINEERING APPROACHES TO NEURAL COMPUTING

1.4.1 Initial Questions

An engineering approach to problem solving involves incorporating all available and relevant problem information in a structured fashion to formulate a solution. Basic questions that arise are

1. Are ANN techniques suitable, or even applicable, to the problem at hand? Does the problem have one or more solutions?
2. Can we develop or modify useful ANN architectures for the situation and, if necessary, train the ANN (determine parameters)?
3. Are there formal tools and heuristics that may be applied to assess the ANN solution properties? (E.g., what is the computational complexity of the solution procedure?)

1.4.2 Neural Engineering Procedures: Replacing Design with Training

Typically, the process of classical engineering "design" involves the systematic application of scientific and mathematical principles to devise a system that meets a set of specifications. In this sense, design may involve judgment, intuition, and possibly iteration. The process of "training," on the other hand, typically involves some form of teaching to force subsequent system behavior to meet specifications. Quite often, this teaching involves the correction or adjustment of system parameters to make system response in the next iteration or experiment closer to that desired.

Neural engineering replaces classical engineering design with determination of ANN-related solution components, including overall ANN architecture, network topologies, unit parameters, and a learning/training procedure. Although this trade-off may seem straightforward, considerable (neural) engineering *judgment* is required. The existence of a myriad of possible choices in topologies and parameters makes exhaustive-search or brute-force network engineering impractical. Furthermore, as mentioned earlier, the suitability of an ANN solution must be determined.

1.4.3 Procedures for ANN System Engineering

During the design of neural network–based solutions, many questions occur, such as

- Can the network be trained to perform the operation desired, or is there some inherent ambiguity in the problem that makes solution impossible?
- Assuming that the problem is solvable, what network structure or topology is appropriate?
- What kind of computing resources are available (time, memory, storage, processors) to train and implement the network?

In realistic applications, the design of an ANN system is a complex, usually iterative and interactive task. Although it is impossible to provide an all-inclusive algorithmic procedure, the following highly interrelated, skeletal steps reflect typical efforts and concerns.

The plethora of possible ANN design parameters include

1. Interconnection strategy/network topology/network structure
2. Unit characteristics (may vary within the network and within subdivisions of the network, such as layers)
3. Training procedure(s)
4. Training and test sets
5. Input/output representation(s) and pre- and postprocessing

A basic design process might proceed as follows:

Step 1: Study the classes of measurements/patterns under consideration to develop possible (hopefully quantitative) characterizations. This includes assessments of (quantifiable) structure, probabilistic characterizations, and exploration of possible class similarity/dissimilarity measures. In addition,

possible deformations or invariant properties and characterization of "noise" sources should be considered at this point.

Step 2: Determine the availability of measurement (input) or feature (preprocessed) data.

Step 3: Consider constraints on desired system performance and computational resources.

Step 4: Consider the availability and quality of training and test data.

Step 5: Consider the availability of suitable and known ANN system structures.

Step 6: Develop an ANN simulation.

Step 7: Train the ANN system.

Step 8: Simulate ANN system performance using test set(s).

Step 9: Iterate among the preceding steps until the desired performance is achieved.

1.5 ANNs: THE MAPPINGS VIEWPOINT

The term *mapping* has several connotations in ANN system design and analysis. The mapping of states in a conceptual problem to ANN states is one example. In the context of the change or mapping of a specific ANN state, a specific input/output (I/O) or perhaps time-varying state mapping may be desired, as shown in Figure 1.2.

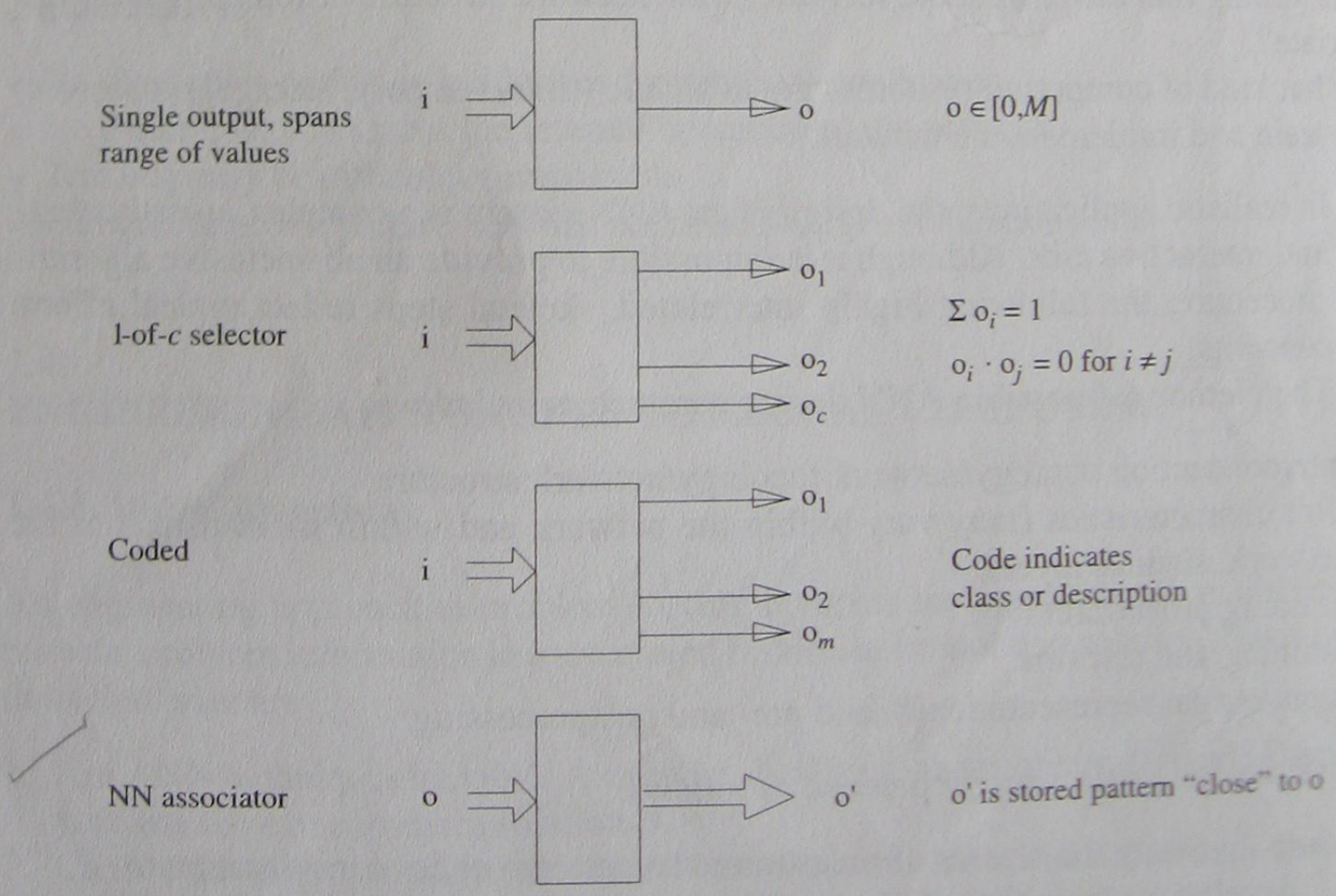


FIGURE 1.2
ANN mapping strategies and input/output representations.

1.5.1 The Basic Perceptual System and Stimulus-Response Approaches

Human perceptual systems include memory and sensory information processing such as visual preprocessing and perception, and auditory preprocessing and perception. The most elemental stimulus-response (S-R) characterization of a perceptual system is shown in Figure 1.3. In biological systems (specifically vertebrates), input from the external world is obtained via *receptor cells,* which respond to a variety of stimuli, including light, heat, chemicals, and mechanical vibration and movement. The input may be a visual pattern, sound waveform, or other biological stimulus. The desired (or learned) response may be a recognition or reaction.

A black-box system is specified by an S-R characteristic. Typically, the internal computation is irrelevant, is not understood, or defies quantification. One viewpoint is that ANNs represent a *nonalgorithmic, black-box computational strategy,* which is trainable. We hope to "train" the neural black box to "learn" the correct response or output (e.g., classification) for each of the training samples. This strategy is attractive to the system designer since the required amount of a priori knowledge and detailed understanding of the internal system operation is minimal. Furthermore, after training, we hope that the internal (neural) structure of the artificial implementation will *self-organize* to enable extrapolation when faced with new, similar patterns on the basis of "experience" with the training set. With the black-box perspective, in the analysis of an application, the question "What's the model?" may receive the trite answer "Who cares?"

The structure of ANNs is often hierarchical. In the black-box sense, this spawns a "box of boxes" structure, where the internal boxes may have a different topological structure. This structure also suggests that we can rearrange the internal boxes (and their contents) and interconnect other "macro-boxes" to achieve new network structures.

The key aspect of black-box approaches is developing relationships between input and output. The adage "garbage in, garbage out" holds for black-box approaches. The success of the approach is likely to be strongly influenced by the quality of the training data and algorithm. Furthermore, the existence of a training set and a training algorithm does not guarantee that a given ANN network will train for a specific application.

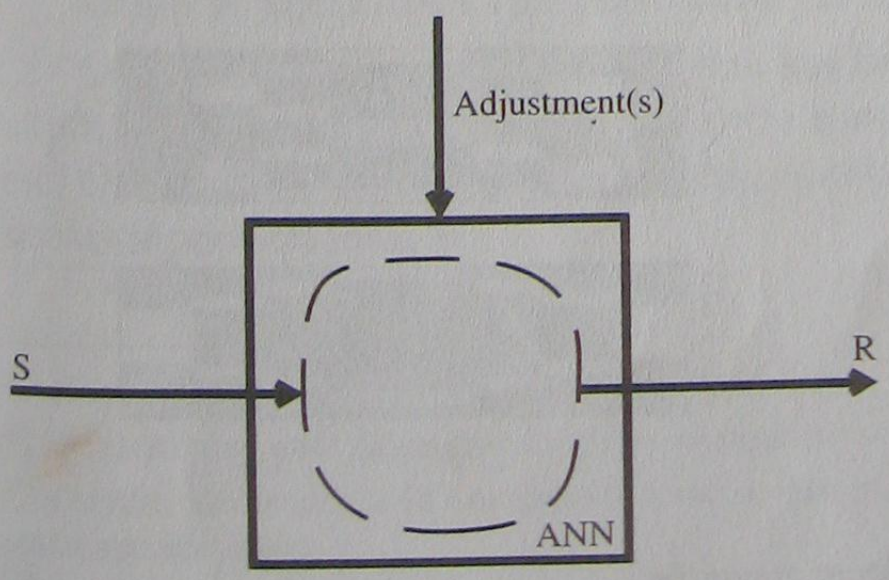


FIGURE 1.3
Basic structure of a "black box" S-R system.

1.5.2 Network Inputs and Outputs

ANN implementations range from situations where, for example, the inputs to individual neurons correspond to values of pixel intensities in an input image, to cases where groups of neurons are used to represent the values of certain features of an object.

This important process is illustrated in the following discussion using a simple character recognition application. Referring to the network character or digit data of Figure 1.4, suppose the objective is to develop an ANN structure with outputs of the form of the second structure in Figure 1.2, i.e., a 1-of-10 selector. For example, when the network is presented with the digit 6, output number 6, denoted o_6, is 1 and $o_j = 0$, $j \neq 6$. Prior to selection of ANN inputs, we note that the digits shown in Figure 1.4 are represented graphically using an 11×8 character box or mathematically as 11×8 binary matrices.

Input selection

Input selection is the process of choosing inputs to the ANN and often involves considerable judgment. Inputs may be preprocessed stimuli, such as quantized and filtered speech data. In some cases there are mathematical tools that help in input selection. In other cases simulation may aid in the choice of appropriate inputs. Clearly, restrictions on measurement systems for a given application may restrict the set of possible inputs. Also, the amount of necessary preprocessing may influence the inputs chosen.

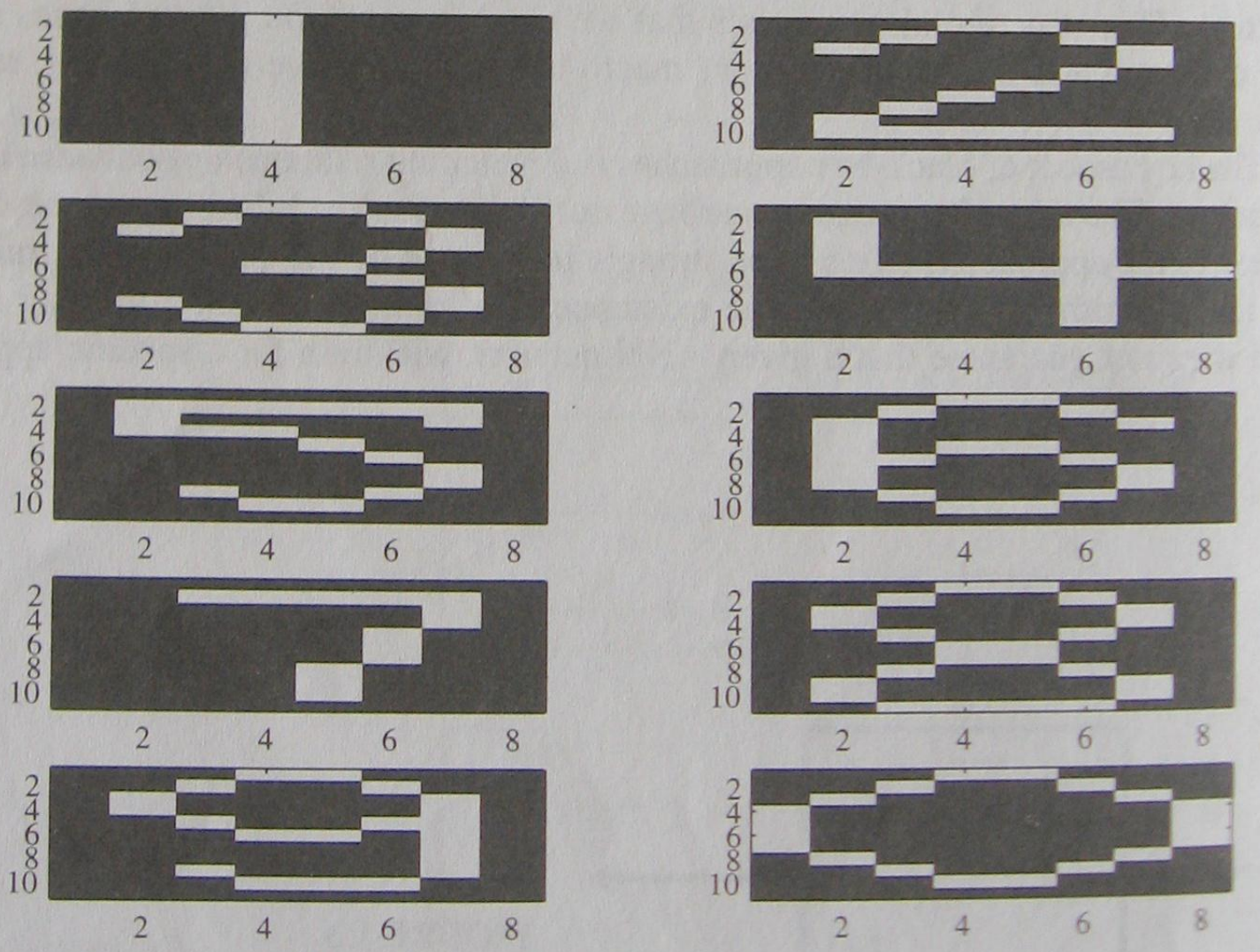


FIGURE 1.4
Digits used to illustrate ANN input/output concerns.

In the aforementioned digit recognition example, one (obvious) choice would be to convert each 11×8 binary matrix into an 88×1 column vector[5] and use this as input to the ANN. Alternatively, following some analysis, features could be extracted from these matrices, at some computational cost, and used as ANN input.[6] At the outset, it is not clear which approach is preferable.

Input distortions

Often, ANN mapping of inputs to outputs that are invariant to some (known) changes or deviations in the input patterns are desired. Thus, the ANN exhibits *invariance* to some input perturbations. An alternative viewpoint is that the same response is desired for a class or set of input stimuli. These deviations may be due to a variety of causes, including "noise." For example, humans are able to recognize printed or handwritten characters with widely varying font sizes and orientations. The exact mechanism that facilitates this capability is unknown.

Again in reference to the digit recognition example, it might be desirable to recognize the digits when small portions are missing or when extra information is present. Furthermore, perhaps recognition independent of position in the 11-character box is desired. These are often difficult objectives to achieve. Note also that the choice of features as input to the ANN may facilitate this.[7]

Output selection

Output selection concerns parallel those of input selection and are almost always application (problem) dependent. As an alternative to the 1-of-10 structure, suppose the ANN output was also an 88×1 vector, corresponding to an 11×8 matrix. The ANN mapping desired should convert noisy or distorted digits to "standard" or reference versions such as those shown in Figure 1.4.

Another example related to the digit recognition problem might be to use the ANN to tell whether the input digit is <5 or ≥ 5. Again using the second structure of Figure 1.2, a 1-of-2 selector or a single binary output is possible. Another design approach might be to develop an ANN that maps the input digits (or features) onto the 1-of-10 structure. A second ANN could then use this output as input and provide the <5 or ≥ 5 output. This structure is shown in Figure 1.5.

A very interesting question is whether this cascaded network has anything in common with a network that does the overall mapping directly.[8]

Other input/output representation issues

Once the problem of input selection has been solved, the choice of input representation becomes paramount. Inputs may be continuous over an interval, discrete, coded, etc. Quite often, the choice of input representation has a strong influence on resulting network performance.

---

[5]Perhaps by row or column concatenation of the matrix.

[6]The reader may wish to suggest some appropriate or reasonable features.

[7]As before, the reader is left to suggest features that might be position invariant yet convey information about specific digits.

[8]The reader may wish to explore this concept along with specific network implementations in later chapters.

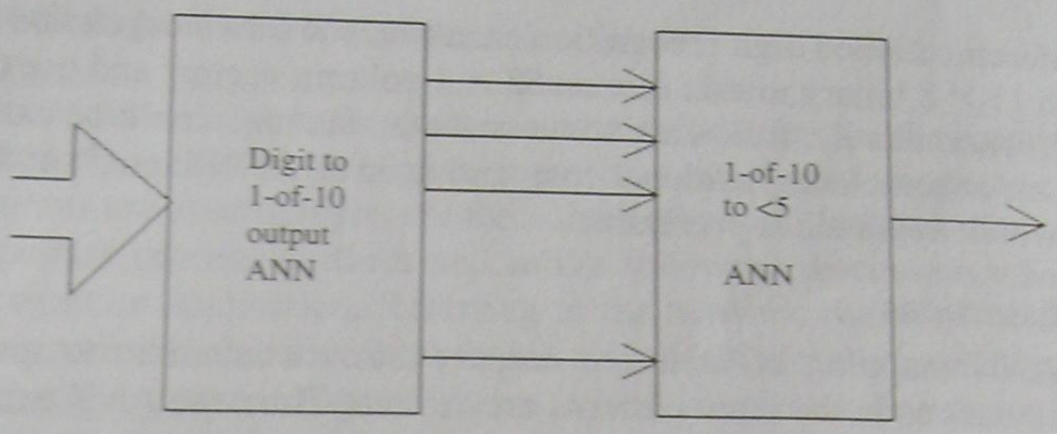


FIGURE 1.5
Use of cascaded ANNs for digit classification.

I/O effect on mappings

The desired behavior of the ANN is characterized as a set of S-R pairs, forming the specification of a relation. For example, ANN behavior as a relation could be characterized by the set of n ordered pairs

$$H\{(s_i, r_i)\} \quad i = 1, 2, \ldots, n \tag{1.1}$$

where s_i is the ith stimulus and r_i the corresponding response. s_i is an element of the relation domain, and r_i is a member of the range. Note that domain s_i and range r_i are expressed in preselected representations; for example, s_i and r_i might be $d \times 1$ and $c \times 1$ vectors, denoted $\underline{s}$ and $\underline{r}$ or $\underline{i}$ and $\underline{t}$, respectively. The goal of the network, given a prespecified topology and unit characteristics, is to *learn the relation* of Equation (1.1). The goal might be to design and verify an ANN that implements the function mapping

$$r_i = f_D(s_i) \tag{1.2}$$

One of the simplest examples of this is when s_i and r_i are real numbers. Equation (1.1) is used to determine the (mapping) function of Equation (1.2); that is, a *curve-fitting* problem results. This is considered in more detail in Section 1.7 and in later chapters.

1.5.3 Vector Representations for S-R Characteristics

Often an ANN stimulus (which may simply be ANN inputs or the current state of all neurons) is represented as a vector $\underline{x}$ and the desired response as $\underline{x}_d$. The desired ANN mapping is then formulated as

$$\underline{f}_D : \underline{s} \rightarrow \underline{r} \tag{1.3}$$

or

$$\underline{r} = \underline{f}_D(\underline{s}) \tag{1.4}$$

One important mapping of the form of Equation (1.4) is *linear mapping*, which may be implemented via a mapping matrix. This is of the form

$$\underline{r} = M\underline{s} \tag{1.5}$$

where the dimensions of M are such that it is conformable to the product shown in Equation (1.5). Finally, the vector formulation of Equation (1.4) does not require stimulus and response vectors to be in the same vector space.

1.5.4 Parameters, Weights, and Constraints

Consider the case of a single unit where we augment the notation to account for unit parameters, i.e.,

$$r = f_p(\underline{s}, \underline{a}_p) \tag{1.6}$$

where $\underline{a}$ represents parameters of unit p and $\underline{s}$ represents the input or stimulus to unit p. Specification of a single desired input/output relation of the form $(\underline{s}, r)$ places an obvious constraint on f_p and $\underline{a}_p$. Additional constraints may be added in the form of other $(\underline{s}, r)$ pairs. Note that the existence of a solution to this constraint satisfaction problem may or may not exist; in fact, there may be multiple solutions. Consider the combined network characteristic

$$\underline{r}_i = \underline{f}(\underline{s}_i, \underline{a}_c, \underline{w}) \tag{1.7}$$

where $\underline{s}_i$ and $\underline{r}_i$ represent network input and output, respectively, $\underline{a}_c$ represents collective network unit characteristics,[9] and $\underline{w}$ represents the network interconnection ("weights"). Again specifying desired network behavior as $(\underline{s}_i, \underline{r}_i)$ pairs yields constraints on $\underline{f}$, $\underline{a}_c$, and/or $\underline{w}$. Most typically, we choose a network structure with some constraints on $\underline{w}$ (such as recurrent) but with initially undetermined interconnection strengths (values of individual weights). In addition, assume that unit characteristics $\underline{f}, \underline{a}_c$ have been chosen. Training then becomes the process of finding one or more solutions (approximate or exact) for $\underline{w}$. This case is explored in considerable detail in Chapters 6 and 8. More general network synthesis, however, is possible.

1.6 ANNs: THE STRUCTURE VIEWPOINT

Any taxonomy to describe ANNs must begin with identification of relevant features. These include

- Unit characteristics
- Learning/training paradigms (software)
- Network topology
- Network function

1.6.1 ANN Functions

The desired behavior of the network provides another approach to distinguishing networks. For example, the desired function of the ANN may be specified by enumeration

[9] These may vary from unit to unit.

of a set of stable network states, or by identifying a desired network output as a function of the network inputs and current state. Popular examples of classifying ANNs by processing objective are

1. *The pattern associator (PA).* This ANN functionality relates patterns, which may be vectors. Commonly, it is implemented using feedforward networks. In Chapters 4–7, this type of network structure is explored in detail. We consider its learning (or training) mechanism and explore properties and nuances of the approach.
2. *The content-addressable memory or associative memory model (CAM or AM).* This neural network structure, best exemplified by the Hopfield model of Chapter 8, is based on ANN implementation of association.
3. *Self-organizing networks.* These networks exemplify neural implementations of unsupervised learning in the sense that they typically self-organize input patterns into classes or clusters based on some form of similarity. Two examples are considered in Chapter 9.

1.6.2 Neural Network Structure

The connectivity of a neural network determines its structure. We looked briefly at recurrent and nonrecurrent structures. Groups of neurons can be locally interconnected to form "clusters" that are only loosely or indirectly connected to other clusters. Alternatively, neurons can be organized into groups or *layers* that are (directionally) connected to other layers. Thus, ANN application requires an assessment of neural network architectures. Possibilities include

1. Designing an application-dependent network structure that performs some desired computation. An example is given in [CG87].
2. Selecting a common preexisting structure for which training algorithms are available. Examples are the feedforward and Hopfield networks.
3. Adapting a preexisting structure to suit a specific application. For example, see [JS88]. This can include the use of *semantics* or other information to give meaning to the behavior of units or groups of units.

Two different "generic" neural network structures are shown in Figure 1.6. These structures are only examples, but they are two that seem to be receiving the most amount of attention.

1.6.3 Network Topologies and Characterization

In viewing network topologies and structures quantitatively as functions of unit interconnections, we can distinguish several concepts:

1. Recurrent networks
2. Nonrecurrent networks

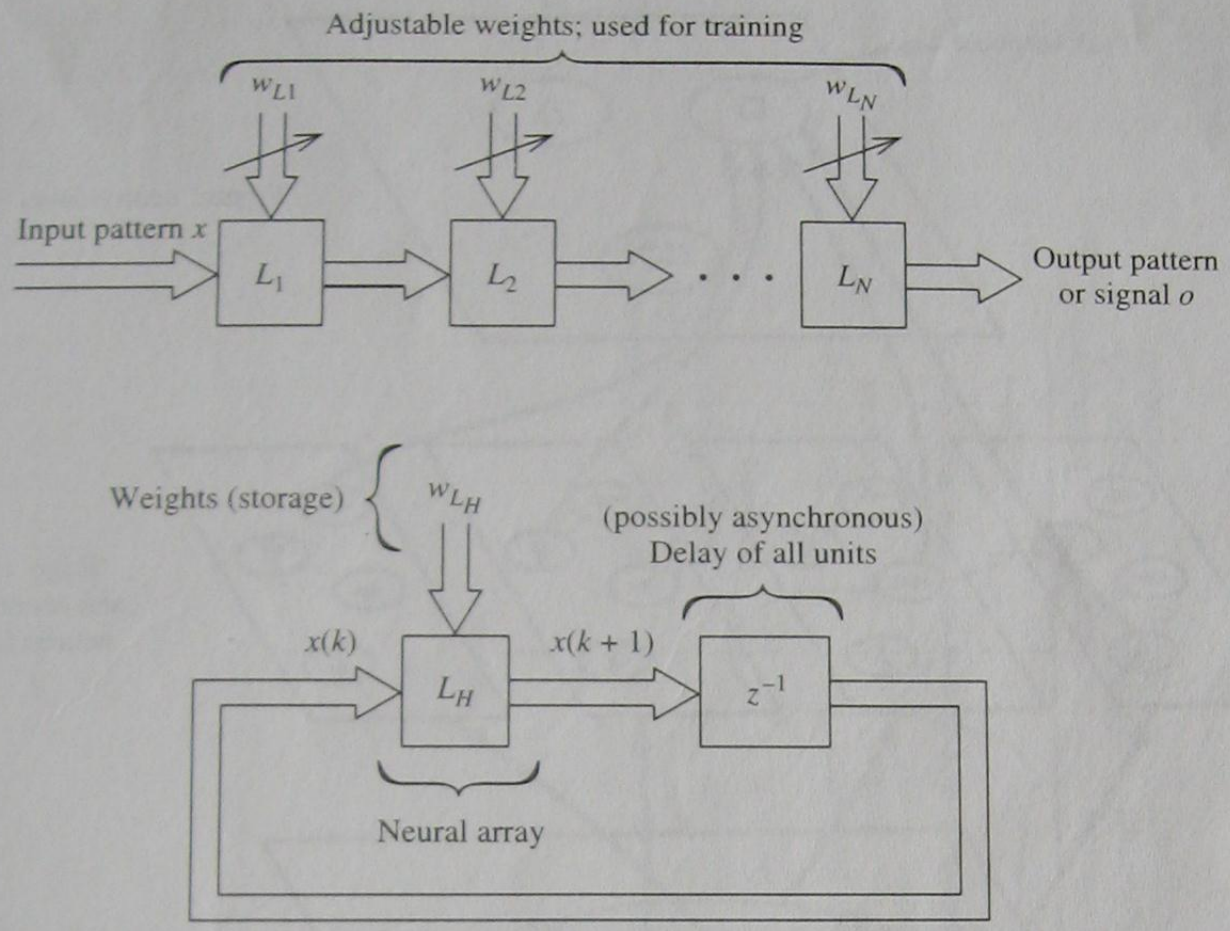


FIGURE 1.6
Generic topological structures.

3. Layered, hierarchical, and other, similarly structured networks
4. Competitive interconnect structures

Types 1 and 2 are mutually exclusive; however types 3 and 4 may apply either to recurrent or nonrecurrent structures. [Fie94] discusses this topic in depth, including the creating of "layers" and "slabs," and distinguishes between symmetric and asymmetric interconnections. Figures 1.7 and 1.8 depict examples of structures of types 3 and 4.

1.6.4 Interconnection Complexity and Problem Scale

The *interconnection complexity* of a network can be significant.[10] This is especially critical when the scaling of the network for larger problems is considered. For example, consider a layered (feedforward) network capable of mapping an $n \times n$ image to another $n \times n$ image. The input and output layers would therefore each require n^2 units. Furthermore, if each input unit were connected to every output unit (totally interconnected layers), the network would yield $(n^2)^2$ or n^4 interconnections. For small n, the total number of interconnections may be insignificant; however, the scaling of the problem raises serious, practical concerns. For example, consider the case of a medium-resolution $n = 512$ image. The corresponding number of interconnections is $n^4 = 6.87 \times 10^{10}$.

[10]This is explored in considerable detail in Chapter 12.

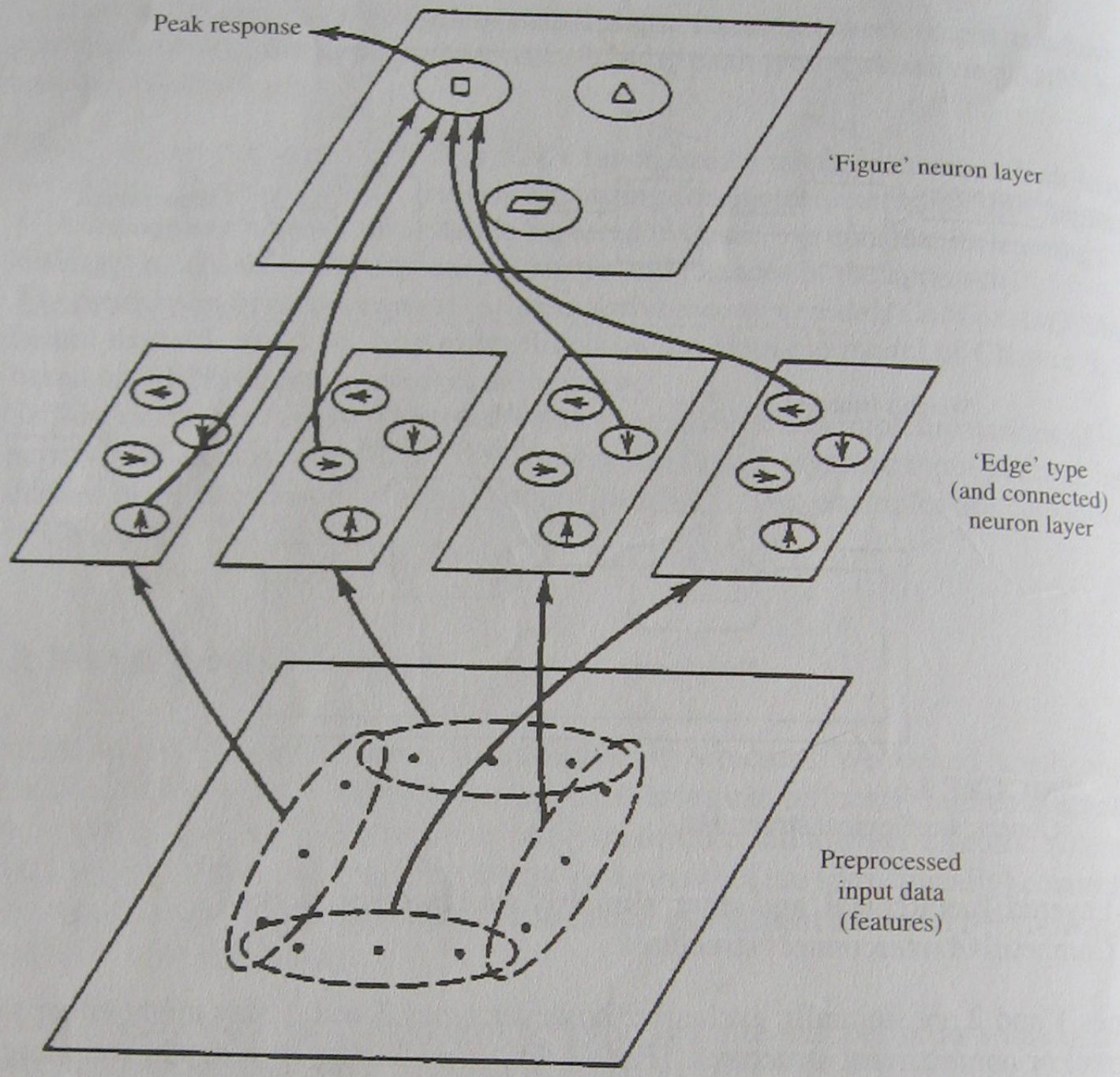


FIGURE 1.7
Sample hierarchical structure of an ANN (vision application).

1.6.5 Feedback Interconnections and Network Stability

The feedback structure of a recurrent network as shown in Figures 1.1 and 1.6 gives rise to network *temporal dynamics*, that is, change over time. In many cases the resulting system, due to the nonlinear nature of unit activation-output characteristics and the weight adjustment strategies, is a highly nonlinear dynamic system. This raises concerns with overall network stability, including the possibility of network oscillation, instability, or lack of convergence to a stable state. The stability of nonlinear systems is often difficult to ascertain.

1.6.6 Combinations of Nets and Variable Topologies

Heretofore we have implied that the ANN designer must design or choose a single network topology. Recent efforts [Has93] suggest that another level of training is to evaluate, and perhaps combine, several topologies for a single application. Although the

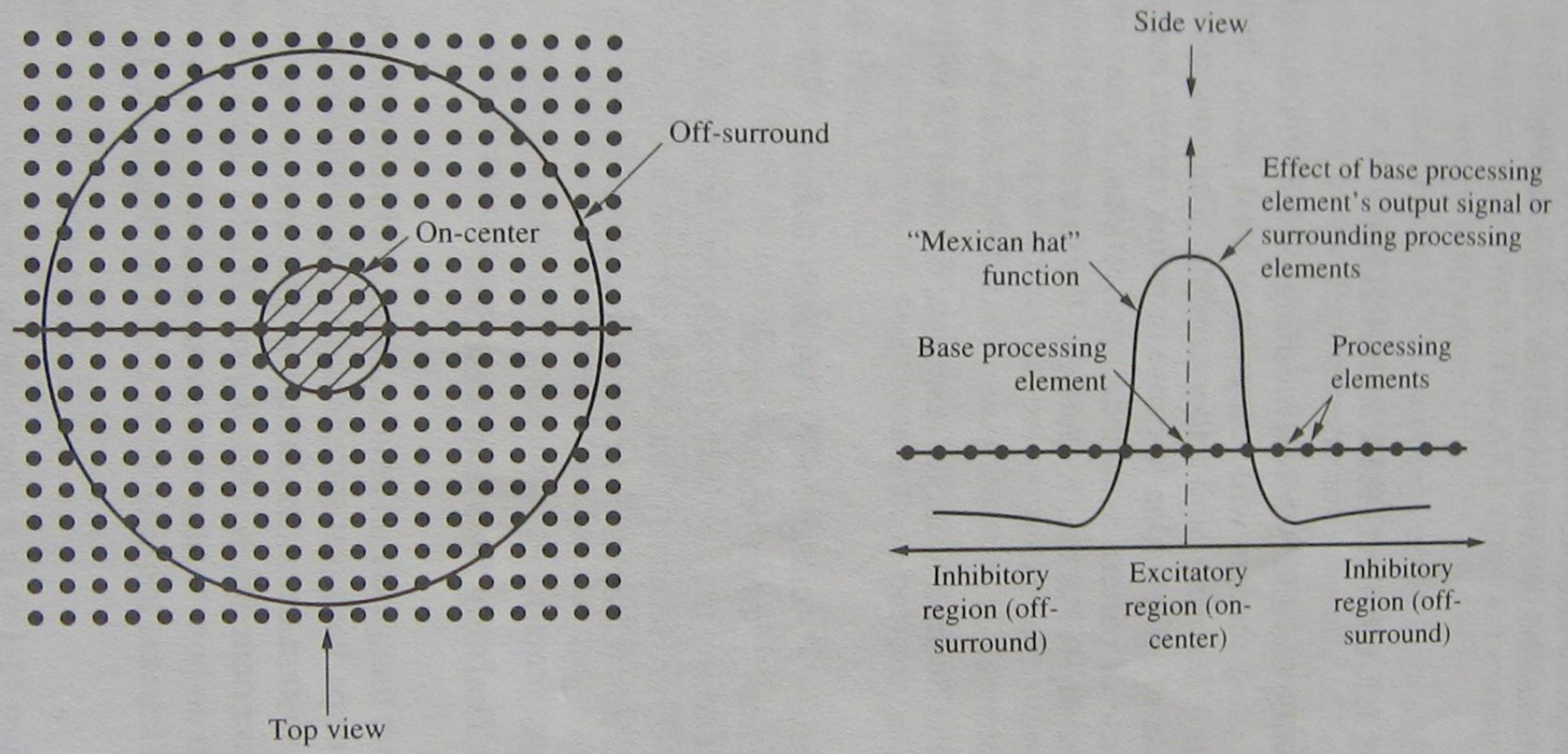


FIGURE 1.8
Sample inhibitory (competitive) ANN structure.

research is in an early stage, the idea is to use optimal (linear) combinations of trained *component* networks. Combinations of component networks form the overall network output, for a specific input, by weighting the outputs of the component networks. The optimization is over a training set of inputs. The training algorithm must select the appropriate emphasis for each component network. Of course, extensions to nonlinear combinations are also worthy of consideration.

1.7 ANN LEARNING APPROACHES

Learning has a very broad meaning [Sch90]. Although the terms are used interchangeably, most readers will recognize at least subtle distinctions between the concepts of training, learning, and understanding. The concept of training may be established as the use of information to cause or refine current mapping behavior, $\underline{f}_A$, toward $\underline{f}_D$. In the case of pattern associator applications, H may be used to iteratively refine $\underline{f}_D$ by comparing the actual response of the untrained system, denoted $\underline{r}_a$, where

$$\underline{r}_a = \underline{f}_A(\underline{s}) \tag{1.8}$$

with the desired or target response, denoted $\underline{r}_d$, where

$$\underline{r}_d = \underline{f}_D(\underline{s}) \tag{1.9}$$

"moving" $\underline{f}_A$ closer to $\underline{f}_D$ through training. Of course, an appropriate measure of "closeness"[11] must be developed.

Learning may be based on *deterministic methods*, such as backpropagation (Chapters 6–7) or Hebbian (Chapter 8) approaches, or *stochastic approaches*, such as genetic algorithms (Chapter 7) or simulated annealing (Chapter 8).

1.7.1 Training Sets and Test Sets

Assume that a certain amount of a priori information, such as sample input/output mappings or perhaps just sample inputs, defining desired system behavior is available to the ANN system designer. In *supervised learning* a set of "typical" I/O mappings forms a database denoted the *training set* (H). In a general sense, H provides significant information on how to associate input data with outputs.

A labeled training set H could be described in the form of ordered pairs:

$$H = \{(\underline{s}_i, \underline{r}_i)\} \quad i = 1, 2, \ldots, n \tag{1.10}$$

Equation (1.10) is a specification for a set of mappings from R^d to R^c. Notice that Equation (1.10) defines only a limited number (n) of the possible infinite number of such mappings. For example, the digits of Figure 1.4 together with the appropriate ANN output for each, as described in Section 1.5.2, could constitute a training set.

---

[11] Vector norm, e.g., $\|\underline{r}_a - \underline{r}_d\|$, is one example.

In *unsupervised* learning, the elements of H are not mappings but rather input or network states, and the ANN must determine "natural" partitions or clusters of the sample data.

An example of supervised learning in a feedforward network is the generalized delta rule (GDR). An example of supervised learning in a recurrent structure is the Hopfield (CAM) approach. Unsupervised learning in a feedforward (nonrecurrent) network is exemplified by the Kohonen selforganizing network. The ART approach exemplifies unsupervised learning with a recurrent network structure.

A mutually exclusive set of additional desired ANN mappings of the form of Equation (1.10) serves as the *test set.* This set is used, following training, to test the generalization capability of the ANN. Referring to the digit recognition example of Section 1.5.2, a test set composed of distorted, noisy, and shifted characters could be used.

1.7.2 Generalization

Any solution to Equation (1.10) in the form $\underline{x}_d = f(\underline{x})$ must satisfy the equation at n points in R^d. An important question arises concerning the behavior of points other than the n $\underline{x}_i$. This introduces the important concept of *ANN generalization* (Chapters 6–7). A generalization example was alluded to in Section 1.5.2, in the discussion of the digit recognition application. The desired generalization was invariance to position and digit distortions such as missing or extra data.

Suppose the desired S-R characteristic for a one-input, one-output ANN is given in the form of a set of four ordered pairs. These points are plotted in Figure 1.9. Both the desired and realized (i.e., after training) generalization capability of this ANN mapping may take many forms. In Figure 1.9, both functional mappings accomplish the objective of mapping the training set (with no mapping error here). Most readers probably prefer the generalization of the curve labeled 1; however, note that the mapping constraint provided solely by the training set allows either solution. Additional (test set) data would be used to test and refine the mapping.

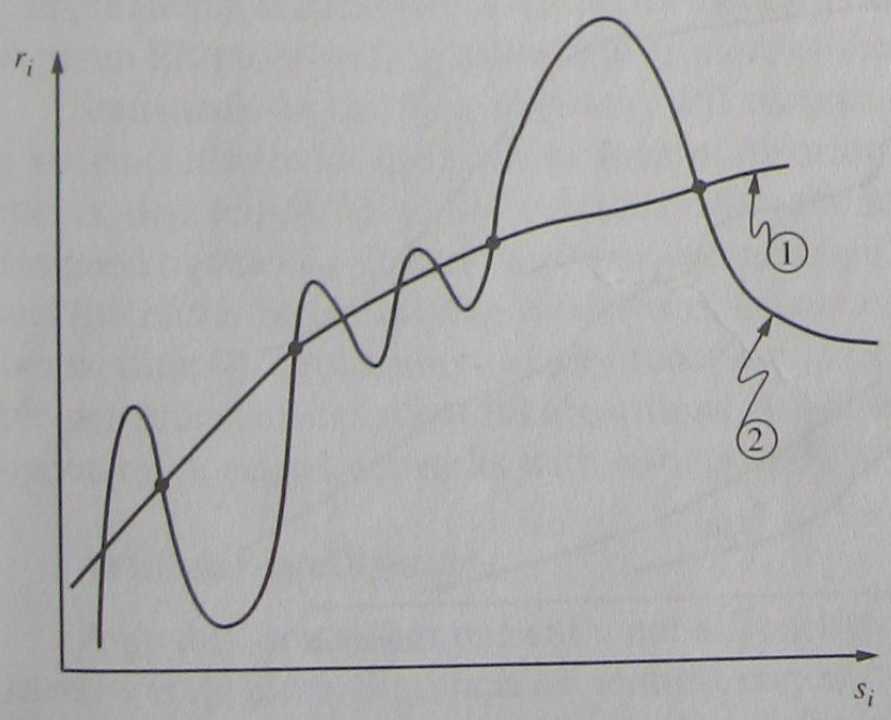


FIGURE 1.9
Generalization versus memorization dilemma shown in the context of curve fitting.

The example of Figure 1.9 yields another interesting and challenging generalization problem. Suppose the inputs and outputs were constrained to be discretized (e.g., integers). How would this constraint be integrated with the ANN design, including through H, to achieve this? In other words, how would we force the ANN generalization to produce integers as opposed to real numbers?

1.7.3 Learning Curves

In areas such as artificial intelligence, learning takes on a more general connotation, somewhat analogous to the self-adaptation processes used by humans [MCM86], ([Sch90], Chapter 18). A learning system may adapt its internal structure to achieve a better response, perhaps on the basis of previously quantified performance. A performance measure could be the difference, or error, between desired and actual system output. This generic learning concept is related to many error correction–based ANN learning techniques, [e.g., the generalized delta rule (GDR) and associated variants in Chapters 6–7]. The GDR is typical of gradient-descent techniques, where the system is modified following each experiment or iteration. This may lead to the typical "learning curve" behavior in biological experiments, where $P(n)$ denotes the probability of the subject (animal or human) making the correct response in the nth trial of a learning experiment. A typical formula [Bol79] for predicting this behavior, which often matches experimental results, is $P(n) = 1 - (1 - P(1))(1 - \theta)^{n-1}$, $n \geq 1$, where $\theta \in [0, 1]$ is a learning parameter. When θ is constrained such that $0 < \theta < 1$, this initial error is reduced in subsequent trials, in a monotonically decreasing manner. Unfortunately, this monotonically increasing performance is often difficult to achieve in practical ANN system training, as typified by Figure 1.10.

Another, related measure is the speed of learning; however, we should not confuse this with the speed of the network in the actual application. Normally, learning is off-line.

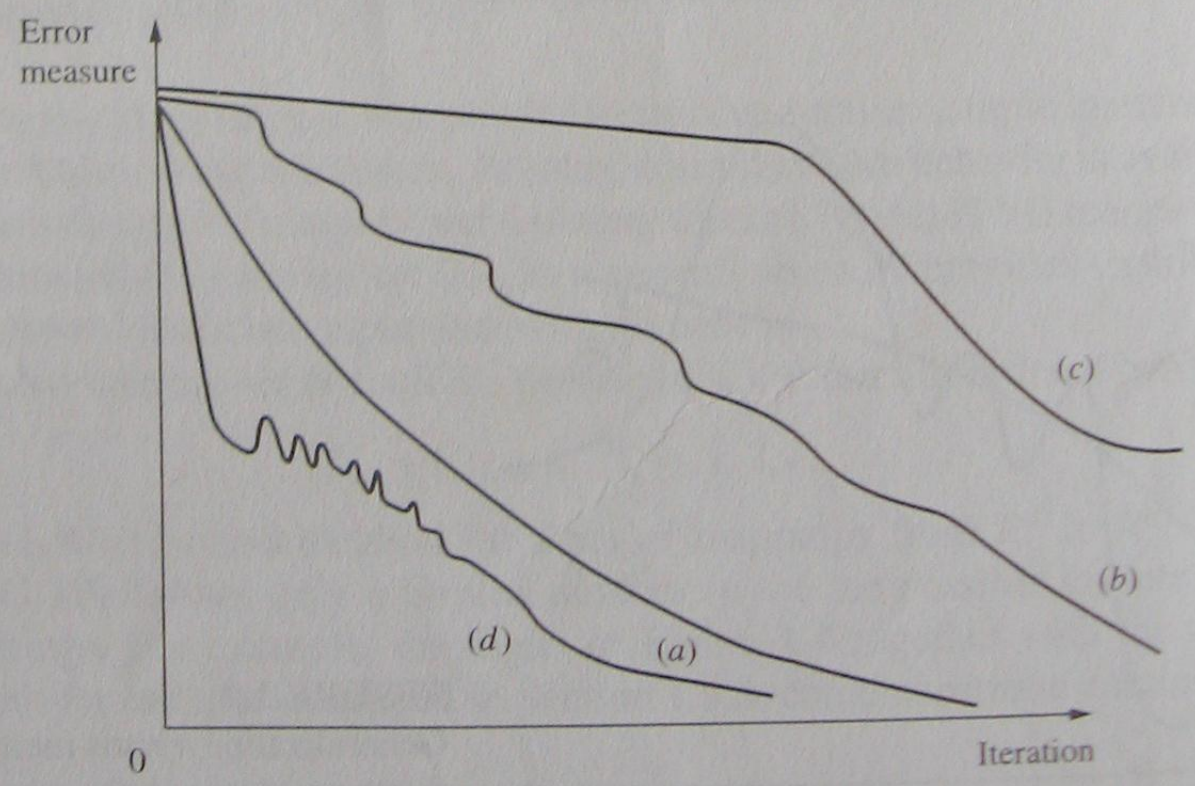


FIGURE 1.10
Sample error measure trajectories.

1.7.4 Error Measures and Error Trajectories

Error measures and error trajectories are generally used to guide and assess training. Various error measures exist, as shown in Chapter 2 and beyond. Typical trajectories are shown in Figure 1.10. Note that Figure 1.10 represents an optimistic scenario in that the error decreases.

1.8 RELATIONSHIP OF ANNs TO OTHER TECHNOLOGIES

Much of what is known about ANN techniques is not new. Furthermore, ANN approaches overlap with other technical areas, such as pattern recognition, computer architecture, (adaptive) signal processing and systems, artificial intelligence, modeling and simulation, optimization/estimation theory, and automata theory.

ANNs are an important component of a related set of technologies often lumped under the moniker "soft computing." Soft computing includes the interrelated areas of neural networks, genetic algorithms, fuzzy systems, pattern recognition, and artificial intelligence. Activity in these areas has been intense; Figure 1.11 shows some of the recent trends.

The flurry of recent publications, near-term application successes, and hype should help to expand this technical discipline.

Pattern recognition

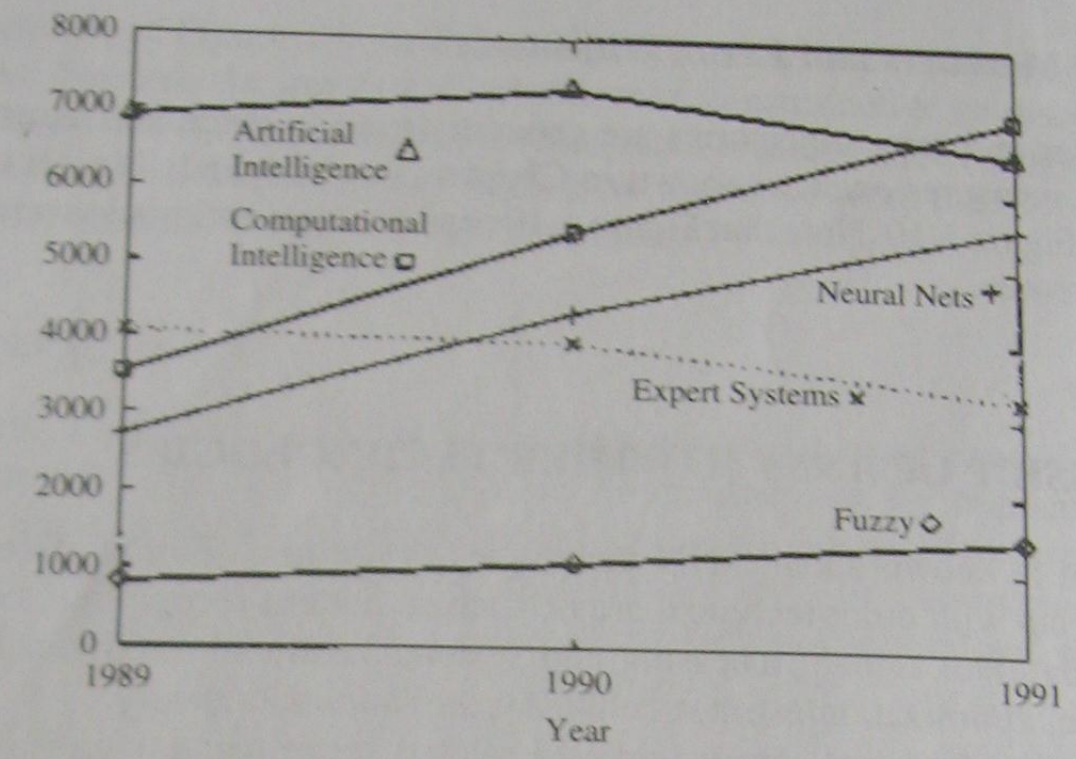
There are important relations between certain aspects of neural computing and the field of pattern recognition. Although these are explored in detail in [Sch92], we briefly summarize them here.

Pattern recognition (PR) applications take many forms. In some cases, there is an underlying and quantifiable *statistical basis* for the generation of patterns. In other cases, the underlying *structure* of the pattern provides the information fundamental for PR. In others, neither of the above cases holds, but we are able to develop and train computational architecture to correctly associate input patterns with desired responses. A given PR problem may allow one or more of these different solution approaches.

Statistical (or decision-theoretic) PR assumes, as its name implies, that there is a statistical basis for the classification of algorithms. A set of characteristic measurements, denoted *features,* are extracted from the input data and each feature vector is assigned to one of c classes. Features are assumed to be generated by a state of nature, and therefore the underlying model is of a state-of-nature– or class-conditioned set of probabilities or probability density functions. There are many examples of neural implementations of statistical PR algorithms for classification and many similarities in the operation of neural networks with statistical PR algorithms.

Artificial intelligence

Arguably, some (but presently not all) aspects of human intelligence may be emulated by computers. Significant scientific, engineering, and mathematical effort is being expended in capturing the architectural and functional aspects of intelligent behavior.



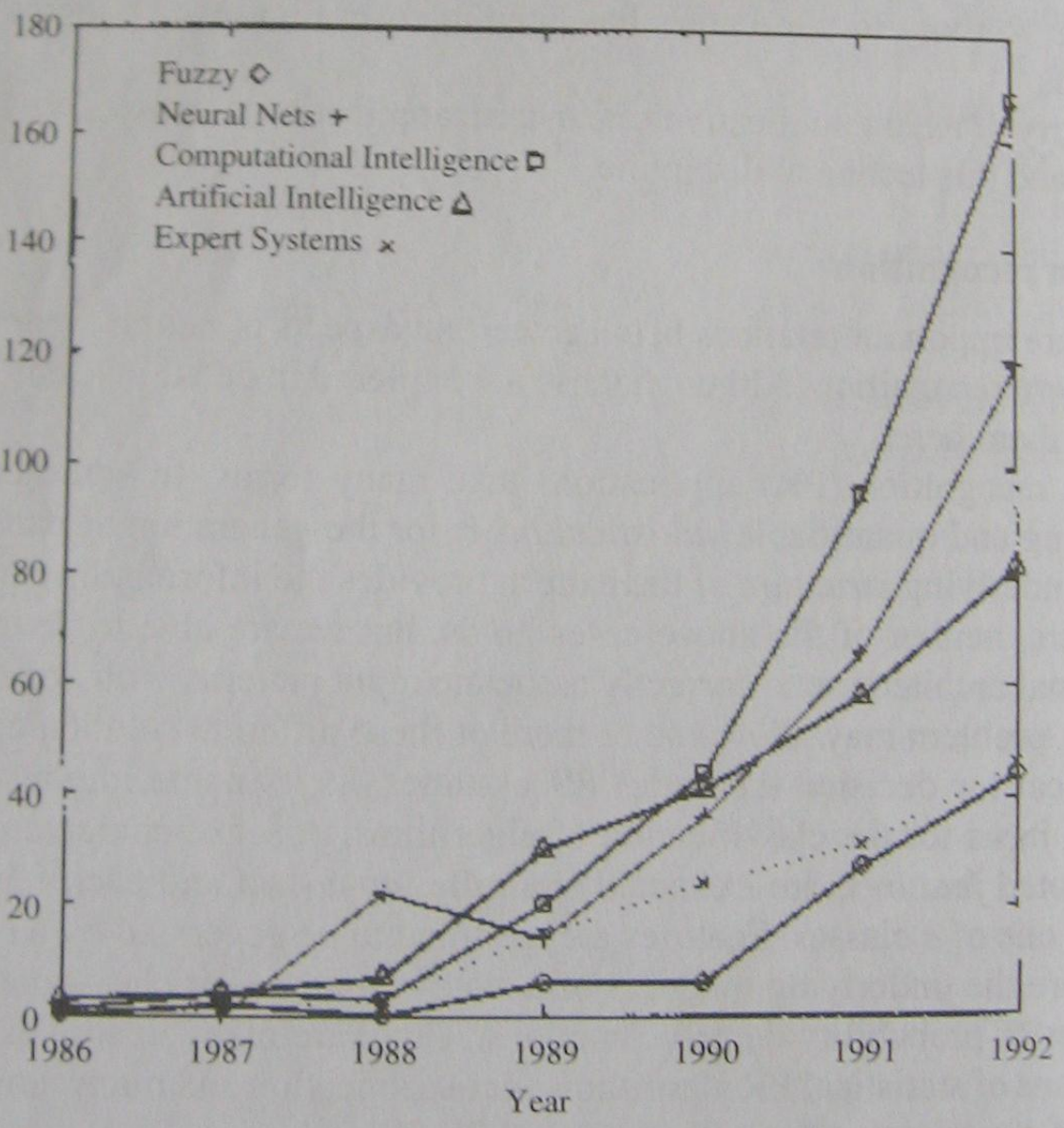


FIGURE 1.11
Recent trends in soft computing efforts.

To this end, the emerging technology of ANNs, or more generally, neurocomputing, could play a major role.

The implementation of expert systems, especially where uncertainty in the inference process must be automated, still poses a significant challenge. To this end, ANNs may contribute in several ways:

- As a means to incorporate uncertainty
- As a means to implement rules or productions
- As a solution to constraint satisfaction and optimization problems

Fuzzy systems

As shown in Chapter 11, ANNs may be used to implement fuzzy systems.

Genetic computing

Genetic computing offers a potentially important mechanism for the training of ANNs. This training is much broader in scope than the simple determination of weights; entire architectures may be searched via genetic algorithms. This is discussed in Chapter 7.

1.9 HISTORICAL EFFORTS

The history[12] of ANN research is not without controversy, and any attempt to summarize it may not meet with uniform approval. The history of ANNs represents the contribution of many researchers. [Nag91] and [SBG87] provide succinct summaries of early ANN research and documentation. We roughly characterize the history of ANNs with the following chronology.

1.9.1 Perceptron and Earlier

The period of the 1940s to the 1960s might be considered the first generation of ANNs. Unfortunately, the networks were not sufficiently complex to solve many interesting problems. Some key events were

McCulloch and Pitts, 1943: Mathematical representation of neuronal processes.

Rosenblatt, 1959: The concept of a single-layer neural network began in the late 1950s at the Cognitive Systems Research Program at Cornell University. In 1957 a technical report was issued that defined Rosenblatt's *perceptron*. The objective was to demonstrate that an adaptive network with massive interconnectivity and nonlinear units could emulate cognitive ability. In 1963 a bibliography on perceptrons listed 98 publications. Although the popular perception is that Rosenblatt considered only single-layer units and binary

[12] It maybe more useful to peruse this section *after* a review of the text, since considerable technical perspective may be necessary to consider these ideas in context.

outputs, this is untrue. His efforts also considered equilibrium conditions for "cross-coupled" and "back-coupled" (recurrent) networks, which were later refined by Hopfield. His final[13] interests concerned the biological basis of learning and associative recall.

Widrow, 1960: The perceptron work was closely paralleled by Widrow's LMS training algorithm for the Adaline/Madaline [WH60] devices. Both of these were single-unit or single-layer machines whose origin was strongly influenced by pattern classification needs. We explore this structure in detail in Chapter 4.

Minsky and Papert, 1969: Minsky and Papert attempted to minimize interest in the perceptron[14] by demonstrating that many of the desirable mappings were unachievable with the perceptron. In this sense, the early history of ANNs concerned supervised training applications. ANN research, although not extinguished, became less active.

1.9.2 Post-Perceptron

In the "second generation" of ANNs, the perceptron shortcomings were addressed via the introduction of general feedforward nets and associated learning algorithms. In addition, new applications, architectures, and training algorithms arose. Arguably, this era was also characterized by the inability to distinguish between a "metaphor" and a "model," as well as by the possible overestimation of the potential abilities of these networks. Some key events were

- Feedforward structures with backpropagation (GDR) training (circa 1985)
- Radial basis function networks
- Hopfield (recurrent) nets (circa 1982)
- BAM (circa 1987)
- Rediscovery and refinement of old concepts, including:

  Hebbian/correlation learning (1949)
  Steinbuch's learning matrix (1963)
  Competitive behavior (1976)

- Adaptive networks

  Grossberg (ART)
  Kohonen (self-organizing)

The post-perceptron era began with the realization that adding (hidden) layers to the network could yield significant computational versatility. This yielded a considerable revival of interest in ANNs (especially multilayered feedforward structures), which continues to this day. The progress occurred on three fronts:

1. The mapping (feedforward) network, when augmented with one or more layers, became able to solve many problems not solvable with the perceptron. On the other

---

[13] Dr. Rosenblatt died in a sailing accident in 1971 at the age of 43.

[14] Some dispute this and simply claim that they were pointing out the limitations of a single-layer device.

hand, suitable algorithms for training the network were necessary, and the success and computational complexity of these algorithms is a topic of continuing research.
2. Other interconnection strategies were considered. The concept of a recurrent network became popular. Some, such as the bidirectional associative memory (BAM) [Kos87], involved recurrent connections of layers, whereas others were simply totally interconnected networks of individual units [Hop82]. The training of recurrent networks became a significant topic, including Hebbian or correlation approaches [Heb49], Steinbuch's learning matrix [SP63], competitive learning [Gro76], and "simulated annealing" [HS86]. A great deal of the early history of learning and mapping strategies is found in [Nil65].
3. Unsupervised learning (truly self-organizing) concepts began to emerge. These include Grossberg- and Kohonen-influenced nets.
4. We should also note that the significant increase in available computing resources that occurred in the two decades following Rosenblatt's work also positively influenced ANN efforts. The computational resources available to Rosenblatt and Widrow are, by today's standards, almost laughable.

1.9.3 The Third Generation of ANNs

The third generation of ANNS, in which performance and practicality issues are paramount, is now upon us. Key issues include

- Assessment of the limitations of networks
- Assessment of the generalization ability of ANNs
- Fusion of ANNs with other technologies, including

  Genetic algorithms
  Fuzzy approaches
- Implementation of ANNs using dedicated hardware

1.9.4 Future Directions and Open Issues

To summarize, we pose a final battery of relatively specific questions:

- What is the complexity of the overall network?
- What is the complexity of the training algorithm?
- How does the ANN solution scale with problem dimensions?
- Does the ANN solution generalize?
- What does the network learn?
- Are parameters other than interconnections (weights) adjusted as part of training?

Answers to these questions will be used to advance the state of the art in ANNs. Specifically, we strive for

1. Better understanding/characterization of ANNs
   - Prediction of future performance (e.g., prove generalization)
   - Prediction or a priori measure of suitability (less trial and error)

2. Better design methodologies and tools, off-the-shelf implementations
3. Incorporation of multiple/independent networks
4. Exploration of comparing/fusing with alternative solutions (e.g., fuzzy systems, genetic approaches)

1.10 OVERVIEW OF ANN LITERATURE AND RESOURCES

1.10.1 Books

The overall field of ANNs is represented by numerous books. Many excellent references exist, and no doubt others are in preparation. Each represents the particular viewpoint and emphasis of the author.

Fundamental neural network architecture and application book references are [AR88], [Pao89], [Kha90], [RM86a], [RM86b], [HN90], and [Kos92]. [Kun93c], [HKP91], [Fau94], and [Zur92] are obviously designed for the student and provide introductory coverage. [Hay94] is quite comprehensive and could easily support a two-semester ANN course sequence. An interesting book, available by ftp, is [KvS93].

Other comprehensive and generally readable references on general neural computing are [Shr88], [Ham93a], [Ham93b], [FFGL88], and [HL87]. Descriptions of 27 artificial neural system paradigms and a brief history of the field are found in [Sim90]. Biological ramifications are considered in [RM86a], [Ros59], and [AR88]. [Was89][15] "provides a systematic entry for the professional who has not specialized in mathematical analysis." Other books include [Kha90], [Sim90], and [CB90].

One of the more interesting reference sources is [WO90]. It is an attempt at a bibliographic guide to ANNs; however, the rate at which change is occurring in this field probably makes it more suitable as a historical reference.

Pattern recognition applications of ANNs are well covered [Sch92]. [Pao89] also attempts to unify the pattern recognition and neural network technologies. The flexibility and geometric complexity of decision boundaries resulting from individual unit characteristics and layering of units is covered in [Lip87] and [MSW90]. [Fu94], [Gal93], and [Lev91] provide an AI viewpoint of ANNs. Hybrid systems, including ANNs, are treated in [GK95]. Digital (only) networks are treated in [Kun93]. Signal-processing applications are emphasized in [CU93]. Financial applications are covered in [IEE91] and [Ref 95]. [HN90] provides Hecht-Nielsen's view of ANNs; [Kos92] provides Kosko's view of ANNs, principally BAM.

Readers interested in dedicated hardware for ANN architectures should consult [Mea89] and [Rot90], as well as the many of the references provided in Chapter 12.

1.10.2 Journals

Work on various aspects of neural computing continues to cross-pollinate journals. Useful sources include

[15] In the words of the author.

IEEE Transactions on Neural Networks (IEEE)
Neurocomputing, the Elsevier Publishing journal, which covers theory, practice, and applications of ANNs
Neural Networks, the official journal of the International Neural Network Society (New York, Pergamon Press)
Neural Computation, published by MIT Press, which disseminates multidisciplinary research results
IEEE Transactions on Systems, Man and Cybernetics
IEEE Transactions on Pattern Analysis and Machine Intelligence
IEEE Transactions on Acoustics, Speech and Signal Processing
Applied Intelligence, the International Journal of Artificial Intelligence, Neural Networks, and Complex Problem-Solving Technologies (Kluwer Academic Publishers)
Neural Computing and Applications, official journal of the Neural Computing Applications Forum (Springer-Verlag)
Journal Of Artificial Neural Networks (Ablex Publishing Company)
International Journal of Neural Systems (IJNS), published quarterly by World Scientific Publishing Co.
Neural Processing Letters, published bimonthly starting in September 1994, with emphasis on ideas, developments, and work in progress

1.10.3 Conferences

Major related periodic conferences include

- IEEE Computer Vision and Image Processing Conference (formerly Pattern Recognition and Image Processing)
- International Conference on Neural Networks (ICNN), sponsored by IEEE

1.10.4 Internet Resources

This important form of ANN resources is rapidly growing and quite volatile. Recently, many sites began to offer hypertext pages on the World Wide Web (WWW).

News groups

Useful discussions[16] are often found in the following USENET news groups:

comp.ai.neural-nets
comp.ai
comp.ai.fuzzy

Sites

Numerous sites dedicated to ANN publication and software continue to appear (and disappear); one of the most stable has been the "Neuroprose" archive at Ohio State, reachable as archive.cis.ohio-state.edu(128.146.8.62) in directory /pub/neuroprose.

[16] And perhaps a lot of ignorance.

Mailing lists

The Neuron Digest is a moderated list (in digest form) dealing with all aspects of neural networks (and any type of network or neuromorphic system). Topics include both connectionist models (artificial neural networks) and biological systems ("wetware"). The digest is posted to comp.ai.neural-nets. Requests to be added to this list should be sent to neuron-request@psych.upenn.edu. Hypertext versions of the Neuron Digest are available via the URL: http://www.erg.abdn.ac.uk/projects/neuralweb/digests/

1.11 OVERVIEW OF THIS BOOK

This text is structured as follows:

- Chapters 1 and 2 introduce the overall subject and provide an overview of the technology. Chapter 2 provides a summary of many important analytic concepts that facilitate the quantitative study of ANNs.
- Chapters 3 to 5 concentrate on the characteristics of single units and simple nets. Chapter 5 explores a special class of relatively simple pattern associators.
- Chapters 6 and 7 present the "feedforward ANN with backpropagation" learning structure, which is quite popular. Chapter 6 introduces the concept; Chapter 7 explores some of the numerous possible extensions.
- Chapter 8 explores the concepts of a highly interconnected dynamic network of nonlinear elements, which is usually attributed to Hopfield.
- Chapter 9 explores several ANNs with unsupervised learning capabilities.
- Chapter 10 explores several extensions to the nets studied earlier, including RBF and TDNN networks.
- Chapter 11 explores ANN applications in the "fuzzy" arena.
- Chapter 12 introduces the important emerging topic of hardware realizations of ANNs.

Following development of the theory, examples are shown to illustrate the concepts. Each chapter contains a concluding section citing relevant literature for more in-depth study of specific topics. Appropriate references are cited.

REFERENCES

[AA82] S. Amari and M. A. Arbib. *Competition and Cooperation in Neural Nets*. Springer-Verlag, New York, 1982 (edited volume).

[AR88] J. A. Anderson and E. Rosenfeld, eds. *Neurocomputing: Foundations of Research*. MIT Press, Cambridge, MA 1988.

[Bol79] Robert Bolles. *Learning Theory*. Holt, Rinehart & Winston, New York, 1979.

[BvdBW94] D. Beastaens, M. van den Bergh, and D. Wood. *Neural Network Solutions for Trading in Financial Markets*. Pitman, Philadelphia, 1994.

[CB90] M. Caudill and C. Butler. *Naturally Intelligent Systems*. MIT Press, Cambridge, MA 1990.

[CG87] G. A. Carpenter and S. Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing,* 37:54–115, 1987.

[CU93] A. Cichocki and R. Unbehauen. *Neural Networks for Optimization and Signal Processing.* John Wiley & Sons, New York, 1993.

[Fau94] Laurene Fausett. *Fundamentals of Neural Networks.* Prentice Hall, Englewood Cliffs, NJ, 1994.

[Fel85] J. A. Feldman. Connectionist models and parallelism in high-level vision. *Computer Vision, Graphics, and Image Processing,* 31:178–200, 1985.

[FFGL88] J. E. Feldman, M. A. Fanty, N. H. Goddard, and K. J. Lynne. Computing with structured connectionist networks. *Communications of the ACM,* 31(2):170–187, Feb. 1988.

[Fie94] E. Fiesler. *Neural Network Classification and Formalization.* Volume 16. Elsevier, New York, 1994.

[Fu94] LiMin Fu. *Neural Networks in Computer Intelligence.* McGraw-Hill, New York, 1994.

[Gal93] S. I. Gallant. *Neural Network Learning and Expert Systems.* MIT Press, Cambridge, MA, 1993.

[GK95] S. Goonatilake and S. Khebbal, eds. *Intelligent Hybrid Systems.* John Wiley & Sons, New York, 1995.

[Gro76] S. Grossberg. Adaptive pattern classification and universal recoding. I: Parallel development and coding of neural feature detectors. *Biological Cybernetics,* 23:121–134, 1976.

[Ham93a] Dan Hammerstrom. Neural networks at work. *IEEE Spectrum,* pp. 26–32, June 1993.

[Ham93b] Dan Hammerstrom. Working with neural networks. *IEEE Spectrum,* pp. 26–53, July 1993.

[Has93] S. Hasham. *Optimal Linear Combinations of Neural Networks.* Ph.D. thesis, School of Industrial Engineering, Purdue University, 1993. Technical Report SMS94-4.

[Hay94] Simon Haykin. *Neural Networks: A Comprehensive Foundation.* IEEE Press, New York, 1994.

[Heb49] D. O. Hebb. *The Organization of Behavior.* John Wiley & Sons, New York, 1949.

[Hin89] G. E. Hinton. Connectionist learning procedures. *Artificial Intelligence,* 40:185–234, 1989.

[HKP91] J. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley, Reading, MA, 1991.

[HL87] W. Y. Huang and R. P. Lippmann. Comparison between neural net and conventional classifiers. *Proceedings, IEEE International Conference on Neural Networks,* IV:485–493, June 1987.

[HN90] R. Hecht-Nielsen. *Neurocomputing.* Addison-Wesley, Reading, MA, 1990.

[Hop82] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of the Sciences,* 79:2554–2558, April 1982.

[HS86] G. E. Hinton and T. J. Sejnowski. Learning and relearning in Boltzmann machines. In D. E. Rummelhart and J. L. McClelland, eds. *Parallel Distributed Processing.* Volume 1. MIT Press, Cambridge, MA, 1986.

[IEE91] IEEE Computing Society Press. *Proceedings of the First International Conference on Artificial Intelligence on Wall Street,* New York, 9–11 Oct. 1991. Cat. No. 91TH0399-6.

[Jef90] C. Jeffries. Code recognition with neural network dynamical systems. *SIAM Review,* 32(4):636–651, Dec. 1990.

[JS88] T. A. Jamison and R. J. Schalkoff. Image labelling via a neural network approach and a comparison with existing alternatives. *Image and Vision Computing,* 6(4):203–214, Nov. 1988.

[Kha90] T. Khanna. *Foundations of Neural Networks.* Addison-Wesley, Reading, MA, 1990.

[Kos87] B. Kosko. Adaptive bidirectional associative memories. *Applied Optics*, 26(23):4947–4960, Dec. 1987.

[Kos92] B. Kosko. *Neural Networks and Fuzzy Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1992.

[Kun93] S. Y. Kung. *Digital Neural Networks*. Prentice-Hall, Englewood Cliffs, NJ, 1993.

[KvS93] B. Kröse and P. vande Smagt. *An Introduction to Neural Nets*. 5th ed., 1993. Available from galba.mbfys.kun.nl (or ftp 131.174.82.73)

[Lev91] Daniel S. Levine. *Introduction to Neural and Cognitive Modeling*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1991.

[Lip87] R. P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, 4:4–22, April 1987.

[MCM86] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine Learning*. Volume II. Morgan-Kaufman, Tioga Park, CA, 1986 (edited volume).

[Mea89] C. Mead. *Analog VLSI and Neural Systems*. Addison-Wesley, Reading, MA, 1989.

[MSW90] R. Winter, M. Stevenson, and B. Widrow. Sensitivity of feedforward neural networks to weight errors. *IEEE Transactions on Neural Networks*, 1(1):71–80, March 1990.

[Nag91] G. Nagy. Neural networks—then and now. *IEEE Transactions on Neural Networks*, 2(2):316–318, March 1991.

[Nil65] N. Nilsson. *Learning Machines*. McGraw-Hill, New York, 1965.

[Ott94] Fighting card fraud. *Ottowa Citizen*, June 1, 1994.

[Pao89] Y. H. Pao. *Adaptive Pattern Recognition and Neural Networks*. Addison-Wesley, Reading, MA, 1989.

[PvG90] J. Paul and E. von Goidammer. Neural net applications in medicine. In J. Stender and T. Addis, eds. *Frontiers in Artificial Intelligence and Applications: "Symbols versus Neurons?"* IOS Press, Washington, DC, 1990, pp. 215–231.

[Ref95] A. N. Refenes. *Neural Networks in the Capital Markets*. John Wiley & Sons, New York, 1995.

[RM86a] D. E. Rummelhart and J. L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 2: Psychological and Biological Models*. MIT Press, Cambridge, MA, 1986.

[RM86b] D. E. Rummelhart and J. L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*. MIT Press, Cambridge, MA, 1986.

[Ros59] F. Rosenblatt. *Principles of Neurodynamics*. Spartan Books, New York, 1959.

[Rot90] M.W. Roth. Survey of neural network technology for automatic target recognition. *IEEE Transactions on Neural Networks*, 1(1):43, March 1990.

[RZF94] A. N. Refenes, A. Zapranis, and G. Francis. Stock performance modeling using neural networks: A comparative study with regression models. *Neural Networks*, 7(2):375–388, 1994.

[Sau89] E. Saund. Dimensionality reduction using connectionist networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11:304–314, March 1989.

[SBG87] S. Shrier, R. L. Barron, and L. O. Gilstrap. Polynomial and neural networks: Analogies and engineering applications. In *IEEE First International Conference on Neural Networks*, 1987, pp. II–431–439.

[Sch90] R. J. Schalkoff. *Artificial Intelligence: An Engineering Approach*. McGraw-Hill, New York, 1990.

[Sch92] R. J. Schalkoff. *Pattern Recognition: Statistical, Structural and Neural Approaches*. John Wiley & Sons, New York, 1992.

[Shr88] B. D. Shriver (ed.). Artificial neural systems (special issue). *IEEE Computer*, 21(3), March 1988.

[Sim90] P. K. Simpson. *Artificial Neural Systems.* Pergamon Press, Elmsford, NY, 1990.

[SP63] K. Steinbuch and V. A. W. Piske. Learning matrices and their applications. *IEEE Transactions on Electronic Computers,* EC12:846–862, Dec. 1963.

[Was89] P. D. Wasserman. *Neural Computing.* Van Nostrand Reinhold, New York, 1989.

[WH60] B. Widrow and M. E. Hoff. Adaptive switching circuits. *1960 IRE WESCON Convention Record,* Part 4, Aug. 1960, pp. 96–104. (Reprinted in [AR88].)

[WO90] P. D. Wasserman and R. M. Oetzel. *NeuralSource: The Bibliographic Guide to Artificial Neural Networks.* Van Nostrand Reinhold, New York, 1990.

[Zur92] J. M. Zurada. *Artificial Neural Systems.* West Publishing, St. Paul, MN, 1992.