FEEDFORWARD NEURAL NETS[1]

Feedforward artificial neural networks and their design algorithms have provided the engineering and statistics com-

[1] Reprinted by permission of Springer-Verlag New York, Inc. from *Feedforward Artificial Neural Networks,* by Terrence L. Fine. Copyright ©1998 by Springer-Verlag New York, Inc.

munities with an effective methodology for the construction of truly nonlinear systems accepting large numbers of inputs and achieving marked success in applications to complex problems of signal processing, pattern classification, and regression/forecasting. Neural networks are adapted to applications through a computer-intensive methodology of learning from examples that requires relatively little prior knowledge, rather than through incorporation of expert beliefs or heuristic programs that require the system designer to understand the essentials of the application. Of course, some prior understanding of the nature of the application is necessary and is reflected in the choice of input variables and network architectures. However, as with other statistical nonparametric methods, we do not need to assume a specific probability model (e.g., multivariate normal). Usually little more is assumed than that the data (training set) is comprised of independent pairs of network inputs (feature vectors) and desired outputs (targets). Our goal is to delineate the capabilities of these systems and to outline the methodology by which they are deployed to confront applications.

In other directions that will not concern us beyond these few words, the network models and their properties illuminate issues in cognitive psychology, neuroscience, and philosophy of knowledge. Network architectures are loosely modeled on, and motivated by, those of the human brain in which (a) the brain itself or a selected functional area (e.g., vision, hearing, balance) is the neural network and (b) the elements/nodes are individual neurons, accepting generally many inputs at dendrites, emitting outputs at axons, and connecting outputs to inputs of many other neurons at unidirectional junctions called synapses. Neuroscience or neurobiology concerns itself with the actual structure and function of living nervous systems. The philosophy motivating the study of neural-based intelligent systems is referred to as connectionism—that is, intelligent responses emerging from the complexity of interconnections between many simple elements and it is a philosophical species of emergentism (e.g., see Ref. 1). Knowledge is stored in the pattern of connection strengths, and intelligent behavior emerges from the collective action of large numbers of simple/unintelligent elements:

> Computational properties of use to biological organisms or to the construction of computers can emerge as collective properties of systems having a large number of simple equivalent components (or neurons). . . . The collective properties are only weakly sensitive to details of the modeling or the failure of individual devices. . . . A study of emergent collective effects and spontaneous computation must necessarily focus on the nonlinearity of the input-output relationship.
>
> *J. Hopfield in Ref. 2*

Nonpolynomial nonlinearity will be seen to be essential if we are to be able to create increasingly complex systems by adding components/neurons.

An overview of the historical background to artificial neural networks is available in Ref. 3, and easy access to the important original articles is provided in Ref. 4. From the mid-1950s to the early 1960s the dominant figure was Cornell's Frank Rosenblatt, who approached modeling brain function as a mathematical psychologist and reported much of his work in *Principles of Neurodynamics* (5). His outlook is reflected in Ref. 6, p. 387:

> The perceptron is designed to illustrate some of the fundamental properties of intelligent systems in general, without becoming too deeply enmeshed in the special, and frequently unknown, conditions which hold for particular biological organisms.

The 1980s saw the reemergence of the study and applications of artificial neural networks initiated by the work of Hopfield (2,7) and the Parallel Distributed Processing Group, reported in Ref. 8, and others (e.g., Amari and Widrow). Hopfield introduced the so-called Hopfield net, also referred to as a feedback or recurrent network, and demonstrated, by relating its behavior to the statistical mechanics study of spinglasses, that these networks could be designed to function as associative or content addressable memories. In the presentation of these networks as real analog circuits, a departure was initiated from thinking of network node functions only as binary-valued discontinuous step functions. A contribution of the PDP group was to show that if you replaced step function nodes by smooth, monotonically increasing but bounded functions, then steepest descent algorithms, particularly as implemented by backpropagation, provided the hitherto missing effective training or learning procedure for complex neural networks. The success of this approach was dependent critically upon the availability in the 1980s of cheap, powerful workstations and personal computers. Large-scale nonlinear optimization problems could now be carried out routinely even when they involved tens of megaflops of computation. In the 1990s the excesses of earlier claims for neural network methods were moderated, much was learned about the capabilities and limitations of neural networks, and sober, encouraging experience was gained from a variety of applications. It is this still-evolving position that we undertake to describe.

NEURAL NETWORK ELEMENTS AND NOTATION

For purposes of initial orientation, we assert that artificial neural networks are networks or systems formed out of many, highly interconnected nonlinear memoryless computing elements or subsystems albeit biological neurons have refractory periods and therefore have memory. The pattern of interconnections can be represented mathematically as a weighted, directed graph in which the vertices or nodes represent basic computing elements, the links or edges represent the connections between elements, the weights represent the strengths of these connections, and the directions establish the flow of information and more specifically define inputs and outputs of nodes and of the network; this is shown in Fig. 1, with the notation of the subsection entitled "Notation" and in the detail required to explain training in the section entitled "Training: Backpropagation."

The pattern of interconnections considered without specification of the weights is referred to as the neural network *architecture.* We will gain an understanding of the capabilities of a feedforward neural network, also called a multilayer perceptron, a network in which the directed graph establishing the interconnections has no closed paths or loops. Such networks will be seen to have significant computational powers but no internal dynamics. In the interests of a coherent development, we will restrict our attention to those networks having several (denoted by d) real-valued inputs, generally denoted by

$$\underline{x} = \{x_1, \ldots, x_d\} \in \mathbb{R}^d$$

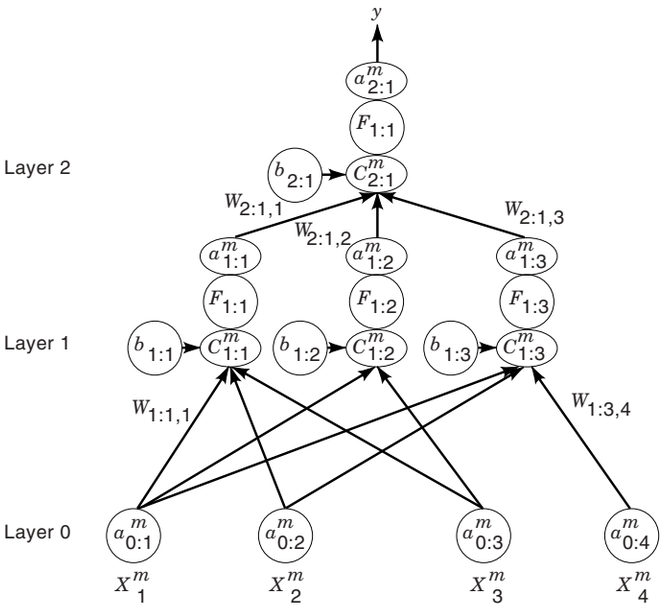

Figure 1. Notation for feedforward neural networks.

Results that are specialized for, say, Boolean-valued inputs ($\underline{x} \in \{0, 1\}^d$) will not be treated. Such results are available from discussions of circuit complexity [e.g., Roychowdhury et al. (9), Siu et al. (10)] and the classical synthesis of Boolean functions [e.g., Muroga (11)].

THE LITERATURE

There has been a sizable literature on the subject of artificial neural networks. While many of the books have been either at a rudimentary level or else edited collections that lack coherence and finish, the last few years have seen the appearance of several substantial books (12–17) and journals devoted to neural networks (18–21). Conference proceedings are plentiful and the most carefully referred is Ref. 22.

ROLES FOR ARTIFICIAL NEURAL NETWORKS

Computation

The brain, with a clock rate of only 100 Hz, is capable of perceptual acts (e.g., recognition of faces and words) and control activities (e.g., walking, control of body functions) that are only now on the horizon for computers with 1 GHz clock rates. The advantage of the brain is assumed to be its effective use of massive parallelism, although there are other features of the brain (e.g., the large numbers of different kinds of neurons, chemical processes that modulate neuronal behavior) that are probably also essential to its effective functioning. The artificial neural network architecture is inherently massively parallel and should execute at high speed—if it can be implemented in hardware. In a multilayer network there is the delay encountered in feeding a signal forward through the individual processing layers. However, this delay is only the product of the number of layers, generally no more than three, and the processing delay in a given layer. Processing within a layer should be rapid because it is generally only a memoryless nonlinear operation on a scalar input that is the scalar product of the vector or array of network inputs with an array of connection weights.

Neural networks probably possess the following properties: (a) graceful degradation of performance with component failure and (b) robustness with respect to variability of component characteristics (e.g., see Refs. 23–25). In part as a consequence of the training algorithms through which the network "learns" its task, a network with many nodes or elements in a given layer has a response that is usually not highly dependent upon any individual node. In such a case, failure of a few nodes or of a few connections to nodes should only have a proportionate effect on the network response—it is not an "all or nothing" architecture. This property holds true of the brain, as we know from the death of neurons over the human lifetime and from the ability of humans to function after mild strokes in which a small portion of the brain is destroyed. Of course, it is possible to set up a network having a critical computing path, and in such a case we do not expect graceful degradation.

Admittedly, neural network applications are still most commonly executed in emulations in Fortran or C and occasionally in MATLAB, and these emulations cannot enjoy some of the advantages just enumerated. However, parallel processing languages are becoming available for use with multiprocessor computing environments (e.g., supercomputers with hundreds of processors) and modern operating systems support program processes containing multiple threads that can be run separately on multiprocessor computers. While there are now special-purpose truly parallel hardware implementations of neural networks in VLSI (e.g., see Refs. 26–29) that provide the highest computational speeds, they are as yet insufficiently flexible to accommodate to the range of applications. This situation, however, is changing rapidly.

Current Systems Applications

The strong interest in neural networks in the engineering community is fueled by the large number of successful and promising applications of neural networks to tasks of pattern classification, signal processing, forecasting, regression, and control. Perceptual recognition tasks, at which biological neural networks have been highly successful, typically involve large numbers of input variables (about 10^6 for vision), none of which are individually critical for recognition. Furthermore, there is generally a nonlinear relationship between the input variables and the goals of the recognition task expressed as target variables. Statistical theory has always encompassed, in principle, optimal nonlinear processing systems. For example, the optimality of the conditional expectation $E(\underline{Y}|\underline{X})$ has long been known in the context of the least mean square criterion $\min_f E\|\underline{Y} - f(\underline{X})\|^2$ when we attempt to infer $\underline{Y}$ from a well-selected function f of the data/observations $\underline{X}$. However, there has been little practical implementation of such nonlinear processors and none when the dimension d of the inputs $\underline{X}$ is large compared to unity. Actual implementations have generally been linear in $\underline{X}$, linear in some simple fixed nonlinear functions of the components of $\underline{X}$, or linear followed by a nonlinear scaling. This has been especially true when confronted with problems having a large number of input variables (e.g., econometric models, percep-

tion). In such cases the usual recourse is to use linear processing based upon knowledge of only means and covariances or correlations. Nonlinear processor design usually assumes knowledge of higher-order statistics such as joint probability distributions, and this knowledge is rarely available. Nonparametric, robust, and adaptive estimation techniques have attempted to cope with only partial statistical knowledge, but with only limited success in real applications of any complexity. Neural networks provide us with a working methodology to design nonlinear systems accepting large numbers of inputs and able to proceed solely from instances of input–output relationships (e.g., pairs $\{\underline{x}_i, \underline{y}_i\}$ of feature vector $\underline{x}$ and pattern category $\underline{y}$) alone. Thus, appropriate applications for neural networks are indicated by the presence of many possible input variables (e.g., large numbers of image pixels, time or frequency samples of a speech waveform, historical variables such as past hourly stock prices over a long period of time), such that we do not know a priori how to restrict attention to a small subset as being the only variables relevant to the classification or forecasting task at hand. Furthermore, we should anticipate a nonlinear relationship between the input variables and the variable being calculated (e.g., one of finitely many pattern categories such as the alphanumeric character set or a future stock price). In some instances (e.g., optical character recognition) these neural network systems provide state-of-the-art performance in areas that have been long-studied. Neural networks make accessible in practice what has hitherto been accessible only in well-studied principle. It is characteristic of human sensory processing systems that they accept many inputs of little individual value and convert them into highly reliable perceptions or classifications of their environment. Classification problems well-suited for neural network applications may be expected to share this characteristic. Examples of pattern classification applications include:

1. Classification of handwritten characters (isolated characters or cursive script)
2. Image classification [e.g., classification of satellite photographs as to land uses (30), face detection (31)]
3. Sound processing and recognition [e.g., speech recognition of spoken digits, phonemes, and words in continuous speech (32,33)]
4. Sonar signal discrimination
5. Classification of acoustic transients, seismic data interpretation
6. Target recognition

Neural-network-based approaches have achieved state-of-the-art performance in applications to the pattern classification problem of the recognition or classification of typed and handwritten alphanumeric characters that are optically scanned from such surfaces as envelopes submitted to the US Postal Service [Jackel et al. (34)], application forms with boxes for individual alphanumeric characters [Shustorovich and Thrasher (35)], or input from a touch terminal consisting of a pad and pen that is capable of recording the dynamical handwriting process [Guyon et al. (36)]. In such applications it is common to have several hundred input variables.

Regression or forecasting problems also confront us with a variety of possible variables to use in determining a response variable such as a signal at a future time. Particular examples of forecasting in multivariable settings that have been the subject of successful neural network-based approaches include forecasting demand for electricity [e.g., Yuan (37)], forecasting fluctuations in financial markets (e.g., currency exchanges), and modeling of physical processes [e.g., Weigend and Gershenfeld (38)].

Control of dynamical systems or plants requires rapid estimation of the state variables governing the dynamics and the rapid implementation of a control law that may well be nonlinear. Rapid implementation is particularly essential in control because control actions must be taken in "real time." Control that is delayed can yield instabilities and degrade performance. Because there can be many state variables, we need to implement functions with many inputs. Frequently there is little prior knowledge of the system structure and the statistical characteristics of the exogeneous forces acting on the system. Such a situation suggests a role for neural network methodology.

Other applications of neural networks such as associative memories for recall of partially specified states, or combinatorial optimization based upon minimization of a quadratic form in a state vector, can be made using feedback, recurrent, or Hopfield networks. However, these networks and the issues of their dynamical behavior and design are treated elsewhere in this encyclopedia.

MATHEMATICAL SETUP FOR FEEDFORWARD NEURAL NETWORKS

Single-Hidden-Layer Functions

Typically the set $\{\underline{x}\}$ of inputs to a neural network is a subset of the d-dimensional reals (d-tuples of real numbers, a vector of dimension d) and, more specifically, often the d-dimensional cube $[a, b]^d$ with side the interval $[a, b]$. We will only consider networks that have a single output y and that compute a real-valued function; vector-valued functions taking values in $\mathbb{R}^m$ can be implemented by m separate networks, one for each of the m components, albeit with a possibly less efficient use of network resources and potential loss of generalization ability due to the increased number of parameters/weights [e.g., see Caruana (39)]. The mathematical structure of a representation of a function through a single hidden layer network having s nodes is presented in the following.

Definition 1. The class $\mathscr{H}_\sigma$ of functions exactly implementable by a single hidden layer (1HL) feedforward net with a single linear output node and hidden-layer node nonlinearity σ is the linear span of functions of $\underline{x}$ of the form $\sigma(\underline{w} \cdot \underline{x} - \tau)$. Restated, $\mathscr{H}_\sigma$ is the set of functions η of $\underline{x}$ specified by

$$\mathscr{H}_\sigma = \{\eta(\cdot, \underline{w}) : (\exists s)(\exists \alpha_1, \ldots, \alpha_s)(\exists \tau_0, \ldots, \tau_s)(\exists \underline{w}_1, \ldots, \underline{w}_s)$$

$$\eta(\underline{x}, \underline{w}) = \tau_0 + \sum_{i=1}^{s} \alpha_i \sigma(\underline{w}_i \cdot \underline{x} - \tau_i)\}$$

where $\underline{w}$ is a vector listing all of the components required to specify η. A given function $\eta(\cdot, \underline{w}) \in \mathscr{H}_\sigma$ is described by p parameters. In the fully connected case,

$$p = (d + 2)s + 1$$

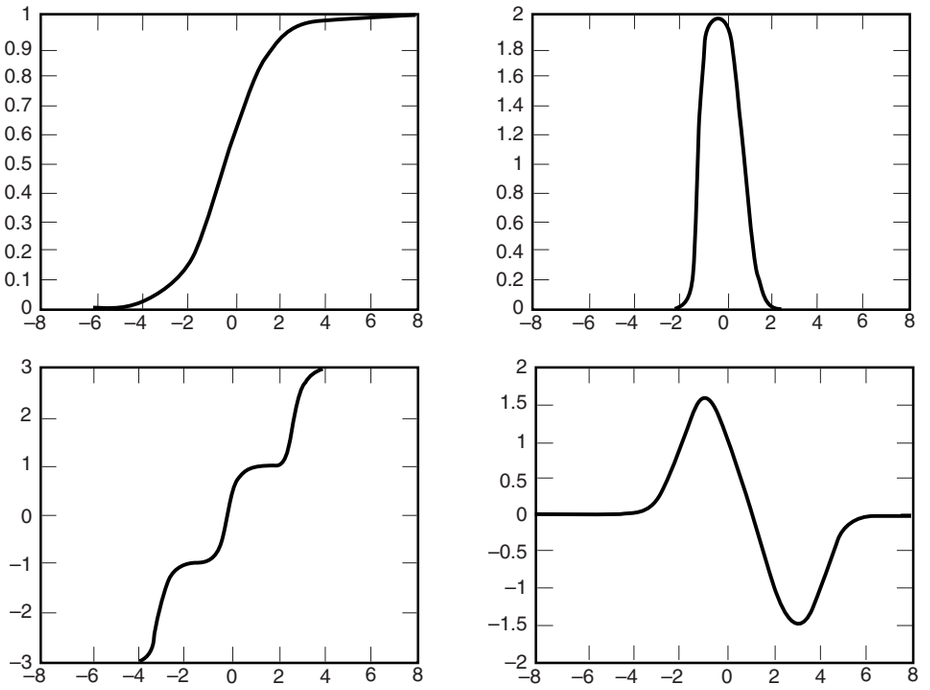

Figure 2. Examples of functions in $\mathscr{H}_\sigma$ for $d = 1$, $s = 1, 2, 3, 4$.

and in practice p is at least several hundred. Examples of functions in $\mathscr{H}_\sigma$, using the logistic node, for $d = 1$ are shown below in Fig. 2. In this figure, nodes are added successively to the previous sum of nodes so as to improve approximation ability by allowing the construction of more complicated functions.

We are interested in the ability of network functions in $\mathscr{H}_\sigma$ to approximate to families of functions of interest. Neural networks can approximate to such important analytical families of functions as the following:

Continuous $\mathscr{C}(\mathscr{X})$. the family of all continuous functions on $\mathscr{X}$

Integrable $L_p(\mathscr{X})$. the family of all Lebesgue integrable pth powers of functions

$$\int_X |f(\underline{x})|^p \, d\underline{x} < \infty$$

rth *Differentiable* $\mathscr{D}_r(\mathscr{X})$. the family of all functions having r continuous derivatives, with $\mathscr{D}_0 = \mathscr{C}$

Piecewise-constant $\mathscr{P}$-constant. the family of all functions on $\mathscr{X}$ that are piecewise constant

Piecewise-continuous $\mathscr{P}$-continuous. the family of all functions on $\mathscr{X}$ that are piecewise continuous

There now exist theorems, one of which we will present below, on the ability of the members of $\mathscr{H}_\sigma$ to approximate to each of the preceding families of functions. Before presenting these results we address some of the elementary mathematical properties of the representations of functions in $\mathscr{H}_\sigma$ and then the meaning we will give to approximation.

Multiple Hidden Layers—Multilayer Perceptrons

Multiple hidden layers enable us to construct compositions of functions in which the outputs of earlier layers become the inputs to later layers. An example of a representation by a two-hidden-layer (2HL) network with a single linear output node is

$$y = \sum_{i=1}^{s_2} w_{3:1,i}\sigma_{2:i}\left(\sum_{j=1}^{s_1} w_{2:i,j}\sigma_{1:j}\left(\sum_{k=1}^{d} w_{1:j,k}x_k\right)\right)$$

As will be seen in the section entitled "The Representational Power of a Single-Hidden Layer Network," single-hidden-layer networks suffice to achieve arbitrarily close approximations for many combinations of family of functions and measure of approximation. However, this is not true for the sup-norm metric (see section entitled "Approximation: Metrics and Closure") and functions that are discontinuous, perhaps being piecewise continuous. A function that is piecewise constant and has hyperplane boundaries for its level sets (regions of constant value) can be constructed exactly using a two hidden layer network, but not always using a single-hidden-layer net [see Gibson (40)]. The need to construct such functions arises in certain inverse problems of control theory, and the resulting controller design may require approximation by a two-hidden-layer network [e.g., see Sontag (41)]. In image and speech recognition problems, multiple-hidden-layer architectures are motivated by attempts to build in spatially or temporally localized features that are expected to be helpful [e.g., see LeCun and Bengio (42)]. For example, in image processing applications, it is common to have a first layer of nodes that are connected only to small, local regions of the image. These node responses are then aggregated in succeeding layers lead-

ing to a final layer of multiple outputs, with each output corresponding to a possible image pattern class.

BASIC PROPERTIES OF THE REPRESENTATION BY NEURAL NETWORKS

Four of the mathematical properties of the representations in $\mathscr{H}_\sigma$ of single-hidden-layer feedforward neural networks, or indeed those with multiple-hidden-layers, are detailed below. They are of value in understanding what happens when we turn to training algorithms.

Multiplicity of Representations

The specification of $\mathscr{H}_\sigma$ is such that different neural networks do not necessarily implement different functions from their inputs in $\mathbb{R}^d$ to, say, a scalar output. Some understanding of this is important for its implications for the existence of multiple approximations of equal quality and its eventual implications for the existence of multiple minima when one comes to training to reduce approximation error. The following are examples of conditions under which distinct networks implement the same function:

1. Permute the nodes in a given hidden layer; for example, interchanging the weights and thresholds corresponding to the first and second nodes will not change the function. In, say, a single-hidden-layer network the network output is $\sum_i \alpha_i \sigma(\underline{w}_i \cdot \underline{x} - \tau_i)$ and the value of the sum is invariant with respect to reordering of the summands.
2. If the node function σ is an odd function, $\sigma(-z) = -\sigma(z)$, such as the commonly employed $\tanh(z)$, then negating the input weights and thresholds to a node and negating the output weights from that same node will introduce sign changes that cancel and leave the network output invariant.
3. If an output weight α_i is 0, then the network response does not vary with changes in w_i.
4. If two nodes in a first hidden layer have the same thresholds and the same connection weights to the network inputs (e.g., $\underline{w}_1 = \underline{w}_2$, $\tau_1 = \tau_2$), then the node responses will be identical. Hence, the weighted summation of their responses $a_1\sigma(\underline{w}_1 \cdot \underline{x} - \tau_1) + a_2\sigma(\underline{w}_2 \cdot \underline{x} - \tau_2)$ in a single-hidden-layer network will depend upon the two output weights only through the sum $a_1 + a_2$ of the two weights.

Albertini et al. (43) and Fefferman (44) examine this issue and provide sufficient conditions under which two networks will implement different functions.

Regions of Constancy

If, as is commonly the case in applications such as character recognition, the number s_1 of nodes in the first hidden layer (and we may allow more than one hidden layer in this discussion) is less than the dimension d of the inputs to the network, then the response of the first hidden layer is determined by s_1 hyperplanes specified by the first-layer individual node weight vectors $\underline{w}_i$ and thresholds τ_i. The locus of constant response from the first hidden layer is then the intersection of these hyperplanes. Equivalently, we can identify the set of input values that share the same function value $\eta(\underline{x}_0, \underline{w})$ as a particular input $\underline{x}_0$:

$$\{\underline{x} : \eta(\underline{x}, \underline{w}) = \eta(\underline{x}_0, \underline{w})\} \supset \cap_{i=1}^{s_1} \{\underline{x} : \underline{w}_i \cdot (\underline{x} - \underline{x}_0) = 0\}$$

If $s_1 < d$, then this intersection is a nonempty linear manifold. Insofar as the first hidden layer responses are constant over this manifold of inputs, all subsequent network responses, including the output, must also be constant over this manifold (and possibly constant over an even larger set).

Stability

It is typically the case for real-valued networks that the selected nonlinear node function σ is continuously differentiable and even analytic in the most common cases of the logistic and hyperbolic tangent functions. For such a choice of node function, the network response $\eta(\underline{x}, \underline{w})$ is also a differentiable function in both its parameters $\underline{w}$ and its arguments $\underline{x}$. Here we consider a 1HL network for which

$$\nabla_{\underline{x}} \eta(\underline{x}, \underline{w}) = \sum_{i=1}^{s_1} \alpha_i \sigma'(\underline{w}_i \cdot \underline{x} - \tau_i) \underline{w}_i$$

is the d-dimensional gradient vector. By the chain-rule of differentiation, η is as many times differentiable with respect to either $\underline{w}$ or $\underline{x}$ as σ is differentiable as a scalar function. Insofar as σ is once continuously differentiable, then we have that $\eta \in \mathscr{D}_1$ and Taylor's Theorem enables us to write

$$\eta(\underline{x}, \underline{w}) = \eta(\underline{x}_0, \underline{w}) + \nabla_{\underline{x}_0}^T \eta(\underline{x} - \underline{x}_0) + o(\|\underline{x} - \underline{x}_0\|)$$

showing a remainder that is of zero order in the difference between input vectors. Thus nearby input vectors give rise to nearby network values—a property of stability of the network representation.

Approximation of Step and Pulse Functions

That feedforward networks enjoy the ability to arbitrarily closely approximate to commonly encountered, well-behaved functions can be motivated by their ability to approximate to step and pulse functions. The most commonly employed nonlinear node functions have the properties of being *sigmoidal* or s-shaped in that they are bounded and nondecreasing and such that

$$-\infty < \underline{\sigma} = \lim_{z \to -\infty} \sigma(z) < \bar{\sigma} = \lim_{z \to \infty} \sigma(z) < \infty$$

For large $a > 0$ and $|z_0|$ such that the product $a|z_0|$ is large enough that $\sigma(\pm a z_0)$ is within a small fraction of its limiting values $\underline{\sigma}$, $\bar{\sigma}$,

$$(\forall z > |z_0|) \frac{|\underline{\sigma} - \sigma(-az)|}{\bar{\sigma} - \underline{\sigma}} < \epsilon, \qquad \frac{\bar{\sigma} - \sigma(az)|}{\bar{\sigma} - \underline{\sigma}} < \epsilon$$

then

$$\frac{\sigma(az) - \underline{\sigma}}{\bar{\sigma} - \underline{\sigma}}$$

is approximately a step function $U(z)$ in $|z| > |z_0|$,

$$\sup_{|z|>|z_0|}\left|U(z) - \frac{\sigma(az) - \underline{\sigma}}{\bar{\sigma} - \underline{\sigma}}\right| < \epsilon$$

If the function f of interest is monotone, then it is easily approximated by a sum of step functions. Recall that a function of bounded variation can be written as a difference of two monotone functions. Hence, the large class of functions of bounded variation are then also easily approximated by sums of step functions. As the functions of bounded variation on a compact set (finite interval) include the trigonometric functions, we see that we can approximate to trigonometric functions and then to those other functions representable as Fourier sums of trigonometric functions.

Given that we can approximate to step functions we can also approximate to pulse functions, with

$$p_\tau(z) = U(z) - U(z - \tau)$$

being a pulse of unit height and width τ. It is clear that we can approximate to common (e.g., continuous) functions by a sum of pulse functions through

$$f(z) \approx \sum_{k=-\infty}^{\infty} f(k\tau)p_\tau(z - k\tau)$$

with accuracy improving with smaller τ. In particular, if f is Lipschitz, then

$$(\exists K)(\forall x, y)|f(x) - f(y)| < K|x - y|$$

Hence,

$$\sup_z\left|f(z) - \sum_{k=-\infty}^{\infty} f(k\tau)p_\tau(z - k\tau)\right| \le K\tau$$

THE REPRESENTATIONAL POWER OF A SINGLE-HIDDEN-LAYER NETWORK

Approximation: Metrics and Closure

In order to present precise results on the ability of feedforward networks to approximate to functions, and hence to training sets, we first introduce the corresponding concepts of arbitrarily close approximation of one function (e.g., the network response) to another function (e.g., the target). To measure approximation error of f by g we consider the distance between them defined by the size of their difference $f - g$, where this is well-defined when both are elements of a normed, linear vector space (e.g., the space of square-integrable functions or usual Euclidean space $\mathbb{R}^d$ with the familiar squared distance between vectors). For the space of continuous functions $\mathscr{C}$ on $\mathscr{X}$ we typically use the metric

$$d(f, g) = \sup_{x\in\mathscr{X}}|f(x) - g(x)|$$

where distance or error here is worst-case error. Another common metric, for $p \ge 1$, is

$$d(f, g) = \left\{\int_{x\in\mathscr{X}}|f(x) - g(x)|^p\mu(dx)\right\}^{1/p}$$

the familiar L_p-norm when the measure μ is the usual Lebesgue measure ($\mu(A) = volume(A)$). This notion of distance provides us with a measure of the degree to which one function approximates another.

The functions in a normed, linear vector space $\mathscr{L}$ that can be arbitrarily closely approximated by ones in a given subset $\mathscr{A} \subset \mathscr{L}$ are the ones in $\mathscr{A}$ as well as the limit points of this set of functions. The set $\mathscr{A}$ together with its limit points is known as the closure $\overline{\mathscr{A}}$ of $\mathscr{A}$ in $\mathscr{L}$ and is specified by the following definition.

Definition 2 (Closure). Given a metric d on a linear vector space of functions $\mathscr{L}$, the closure in $\mathscr{L}$, $\overline{\mathscr{A}}$, of a family $\mathscr{A} \subset \mathscr{L}$ of functions is

$$\overline{\mathscr{A}} = \{g \in \mathscr{L} : (\forall\epsilon > 0)(\exists f_\epsilon \in \mathscr{A}) \quad d(g, f_\epsilon) < \epsilon\}$$

Universal Approximation to Functions

The power of even single-hidden-layer feedforward neural networks is revealed in the technical results cited below. A large number of contributions to this issue have been made, with the major ones first appearing in 1989 [e.g., Refs. 45 and 46]. In essence, almost any nonpolynomial node function used in such a network can yield arbitrarily close approximations to functions in familiar and useful classes, with the approximation becoming arbitrarily close as the width of the layer is increased. That σ not be a polynomial is clearly a necessary condition since a single-hidden-layer network with polynomial nodes of degree p can only generate a polynomial of degree p no matter what the width s of the hidden layer. To report these somewhat technical results we need to define first the set M of node functions.

Definition 3 [Leshno et al. (47)]. Let $M = \{\sigma\}$ denote the set of node functions such that:

1. The closure of the set of points of discontinuity of any $\sigma \in M$ has zero Lebesgue measure (length).
2. For every compact (closed, bounded) set $K \subset \mathbb{R}$, the essential supremum of σ on K, with respect to Lebesgue measure ν, is bounded

$$\text{ess sup}_{x\in K}|\sigma(x)| = \inf\{\lambda : v\{x : |\sigma(x)| \ge \lambda\} = 0\} < \infty$$

For example, property 1 is satisfied if the points of discontinuity have only finitely many limit points, while property 2 is satisfied if σ is bounded almost everywhere. We can now assert

Theorem 1 [Leshno et al. (44), Theorem 1]. Let $\sigma \in M$, then the closure of the linear span of $\sigma(\underline{w}\cdot\underline{x} - \tau)$ is $\mathscr{C}(\mathbb{R}^d)$ if and only if σ is not almost everywhere an algebraic polynomial.

Noting that sigmoidal nodes satisfy the conditions of this theorem, we see that networks composed of them enjoy the ability to universally approximate to continuous functions.

While the preceding theorem tells us much about the power of feedforward neural networks to approximate functions according to specific norms or metrics, there are issues

that are not addressed. For example, we may wish to approximate not only to a function $t(\underline{x})$ but also to several of its derivatives. An approximation, say, using step functions can give an arbitrarily close approximation in sup-norm to a differentiable function of a single variable, yet at no point approximate to its derivative; the approximating function has derivatives that are zero almost everywhere. Results on the use of neural networks to simultaneously approximate to a function and several of its derivatives are provided in Refs. 48 and 49. Results on approximation ability in terms of numbers of nodes have also been developed along lines familiar in nonlinear approximation theory, and these include the work of Barron (50) and Jones (51). They show that in certain cases (in a Hilbert space setting) approximation error decreases inversely with the number s of single hidden layer nodes, and this decrease can in some cases be surprisingly independent of the dimension d of the input.

Universal Approximation to Partial Functions

We now turn to the problem of approximating closely to a partially specified function. The problem format is that we are given a training set

$$\mathscr{T} = \{(\underline{x}_i, \underline{t}_i), i = 1:n\}$$

of input–output pairs that partially specify $t = f(\underline{x})$, and we wish to select a net $\eta(\cdot, \underline{w})$ so that the output $\underline{y}_i = \eta(\underline{x}_i, \underline{w})$ is close to the desired output $\underline{t}_i$ for the input $\underline{x}_i$. This is the typical situation in applications of neural networks—we do not know f but have points on its graph. If instead you are fortunate enough to be given the function f relating $\underline{t}$ to $\underline{x}$, then you can generate arbitrarily large training sets by sampling the function domain, either deterministically or randomly, and calculating the corresponding responses, thereby reducing this problem to the one we will treat in detail in the next section.

The notion of "closeness" on the training set $\mathscr{T}$ is typically formalized through an error or objective function or metric of the form

$$\mathscr{E}_{\mathscr{T}} = \frac{1}{2}\sum_{i=1}^{n} \|\underline{y}_i - \underline{t}_i\|^2$$

Hence, $\mathscr{E}_{\mathscr{T}} = \mathscr{E}_{\mathscr{T}}(\underline{w})$, a function of $\underline{w}$, since $\underline{y}$ depends upon the parameters $\underline{w}$ defining the selected network η. Of course, there are infinitely many other measures of closeness (e.g., metrics such as "sup norm" discussed in the section entitled "Approximation: Metrics and Closure"). However, it is usually more difficult to optimize for these other metrics through calculus methods, and virtually all training of neural networks takes place using the quadratic metric even in some cases where eventual performance is reported for other metrics.

It is apparent from the results of the section entitled "Universal Approximation to Functions" that one can expect a single-hidden-layer network to be able to approximate arbitrarily closely to any given training set $\mathscr{T}$ of size n provided that it is wide enough ($s_1 \gg 1$). An appropriate measure of the complexity of a network that relates to its ability to approximate closely to a training set is given by the notion of Vapnik–Chervonenkis (VC) dimension/capacity. Discussion of VC dimension is available from Vapnik (52), Kearns and Vazirani (53), and Fine (13). Studies of network generalization ability (see section entitled "Learning and Generalization Behavior") also rely on VC dimension.

TRAINING A NEURAL NETWORK: BACKGROUND AND ERROR SURFACE

Error Surface

We address the problem (posed in the section entitled "Universal Approximation to Partial Functions") of selecting the weights $\underline{w}$ and thresholds, generically referred to simply as "weights," to approximate closely to a function partially specified by a training set. We are confronted with the following nonlinear optimization problem:

$$\text{minimize } \mathscr{E}_{\mathscr{T}}(\underline{w}) \text{ by choice of } \underline{w} \in \mathscr{W} \subset \mathbb{R}^p$$

The inherent difficulty of such problems is aggravated by the typically very high dimension of the weight space $\mathscr{W}$; networks with hundreds or thousands of weights are commonly encountered in image processing and optical character recognition applications. In order to develop intuition, it is helpful to think of $\underline{w}$ as being two-dimensional and determining the latitude and longitude coordinates for position on a given portion $\mathscr{W}$ of the surface of the earth. The error function $\mathscr{E}_{\mathscr{T}}(\underline{w})$ is then thought of as the elevation of the terrain at that location. We seek the point on $\mathscr{W}$ of lowest elevation. Clearly we could proceed by first mapping the terrain, in effect by evaluating $\mathscr{E}_{\mathscr{T}}$ at a closely spaced grid of points, and then selecting the mapped point of lowest elevation. The major difficulty with this approach is that the number of required grid points grows exponentially in the dimension of $\mathscr{W}$ (number of parameter coordinates). What might be feasible on a two-dimensional surface will quickly become impossible when we have, as we usually will, a more than 100-dimensional surface.

One expects that the objective function $\mathscr{E}_{\mathscr{T}}(\underline{w})$ for a neural network with many parameters defines a highly irregular surface with many local minima, large regions of little slope (e.g., directions in which a parameter is already at a large value that saturates its attached node for most inputs), and symmetries (see section entitled "Basic Properties of the Representation by Neural Networks"). The surface is technically smooth (continuous first derivative) when we use the usual differentiable node functions. However, thinking of it as smooth is not a good guide to our intuition about the behavior of search/optimization algorithms. Figure 3 presents two views of a three-dimensional projection (two parameters selected) of the error surface of a single node network having three inputs and trained on ten input–output pairs.

Multiple Stationary Points

The arguments of the section entitled "Basic Properties of the Representation by Neural Networks" establish the existence of multiple minima. Empirical experience with training algorithms shows that different initializations almost always yield different resulting networks. Hence, the issue of many minima is a real one. A construction by Auer et al. (54) shows that one can construct training sets of n pairs, with the inputs

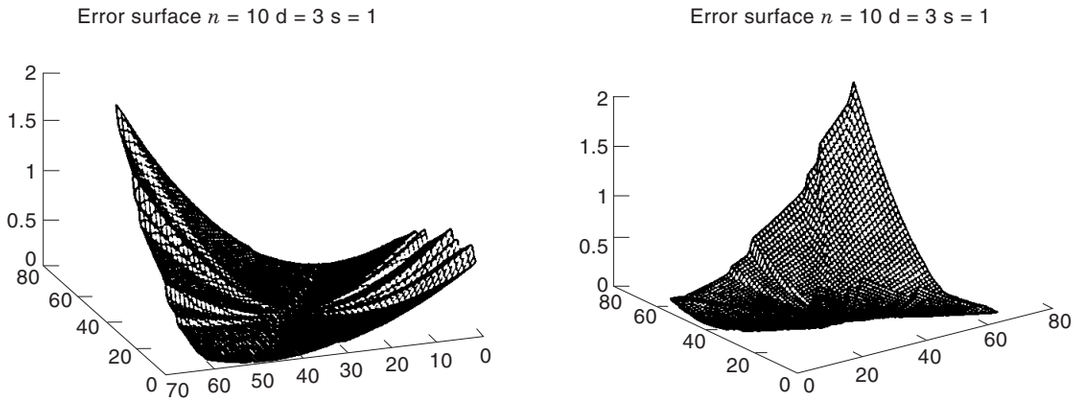

Figure 3. Two views of an error surface for a single node.

drawn from $\mathbb{R}^d$, for a single-node network with a resulting exponentially growing number

$$\left(\frac{n}{d}\right)^d$$

of local minima! Hence, not only do multiple minima exist, but there may be huge numbers of them.

The saving grace in applications is that we often attain satisfactory performance at many of the local minima and have little incentive to persevere to find a global minimum. Recent techniques involving the use of families of networks trained on different initial conditions also enables us, either through linear combinations of the trained networks (e.g., see Refs. 21 and 55) or through a process of pruning, to achieve good performance.

Outline of Approaches

There is no "best" algorithm for finding the weights and thresholds for solving the credit assignment problem that is now often called the *loading problem*—the problem of "loading" the training set $\mathcal{T}$ into the network parameters. Indeed, it appears that this problem is intrinsically difficult (i.e., NP-complete versions). Hence, different algorithms have their staunch proponents who can always construct instances in which their candidate performs better than most others. In practice today there are four types of optimization algorithms that are used to select network parameters to minimize $\mathcal{E}_{\mathcal{T}}(\underline{w})$. Good overviews are available in Battiti (56), Bishop (12), Fine (13), Fletcher (57), and Luenberger (58). The first three methods, steepest descent, conjugate gradients (e.g., Møller (59), and quasi-Newton (see preceding references), are general optimization methods whose operation can be understood in the context of minimization of a quadratic error function. While the error surface is surely not quadratic, for differentiable node functions it will be so in a sufficiently small neighborhood of a local minimum, and such an analysis provides information about the high-precision behavior of the training algorithm. The fourth method of Levenberg and Marquardt [e.g., Hagan and Menhaj (60), Press et al. (61)] is specifically adapted to minimization of an error function that arises from a quadratic criterion of the form we are assuming. A variation on all of the above is that of **regularization** [e.g., Tikhonov (62), Weigend (63)] in which a penalty term is added to the performance objective function $\mathcal{E}_{\mathcal{T}}(\underline{w})$ so as to discourage excessive model complexity (e.g., the length of the vector of weights $\underline{w}$ describing the neural network connections). All of these methods require efficient, repeated calculation of gradients and backpropagation is the most commonly relied upon organization of the gradient calculation. We shall only present the steepest-descent algorithm; it has been the most commonly employed and limitations of space preclude presentation of other approaches.

TRAINING: BACKPROPAGATION

Notation

Formal exposition of feedforward neural networks (FFNN) requires us to introduce notation, illustrated in Fig. 1, to describe a multiple layer FFNN, and such notation has not yet become standard.

1. Let i generically denote the ith layer, with the inputs occurring in the 0th layer and the last layer being the Lth and containing the outputs.
2. A layer is indexed as the first subscript and separated from other subscripts by a colon (:).
3. It is common in approximation problems (e.g., estimation, forecasting, regression) for the last layer node to be linear but to be nonlinear in pattern classification problems where a discrete-valued response is desired.
4. The number of nodes in the ith layer is given by the width s_i.
5. The jth node function in layer i is F_{ij}; alternatively we also use σ_{ij}.
6. The argument of F_{ij}, when $\underline{x}_m$ is the input to the net, is denoted c_{ij}^m.
7. The value of $F_{ij}(c_{ij}^m)$ equals a_{ij}^m when the net input $\underline{x}_m$ equals $\{x_j^m = a_{0j}^m\}$ and the vector of node responses in layer i is a_i.
8. The derivative of F_{ij} with respect to its scalar argument is denoted f_{ij}.
9. The thresholds or biases for nodes in the ith layer are given by the s_i-dimensional vector $\underline{b}_i = \{b_{ij}\}$.

10. The weight $w_{i;j,k}$ assigned to the link connecting the kth node output in layer $i-1$ to the jth node input in layer i is an element of a matrix $\mathbf{W}_i$.

Hence, in this notation the neural network equations are

$$a^m_{0:j} = (\underline{x}_m)_j = x^m_j, \qquad \underline{a}^m_{0:} = \underline{x}_m \tag{1}$$

$$c^m_{i:j} = \sum_{k=1}^{s_{i-1}} w_{i:j,k} a^m_{i-1:k} + b_{i:j}, \qquad \underline{c}^m_{i:} = \mathbf{W}\underline{a}^m_{i:} + \underline{b}_i \tag{2}$$

$$a^m_{i:j} = F_{i:j}(c^m_{i:j}), \qquad \underline{a}^m_{i:} = \mathbf{F}_i(\underline{c}^m_{i:}), \qquad a^m_{L:1} = y_m \tag{3}$$

For clarity we assume that the network has a single output; the extension to vector-valued outputs is straightforward but obscures the exposition. The discrepancy e_m between the network response y_m to the input $\underline{x}_m$ and the desired response t_m is given by

$$e_m = y_m - t_m = a^m_{L:1} - t_m, \qquad \underline{e} = (e_m)$$

and the usual error criterion is

$$\mathscr{E}_m = \frac{1}{2}(y_m - t_m)^2 = e^2_m, \qquad \mathscr{E}_{\mathscr{T}} = \sum_{m=1}^{n} \mathscr{E}_m(\underline{w}) = \underline{e}^T\underline{e} \tag{4}$$

Backpropagation

A systematic organization of the calculation of the gradient for a multilayer perceptron is provided by the celebrated *backpropagation algorithm*. We supplement our notation by introducing w as an enumeration of all weights and thresholds/biases in a single vector and defining

$$\delta^m_{i:j} = \frac{\partial \mathscr{E}_m(\underline{w})}{\partial c^m_{i:j}} \tag{5}$$

To relate this to our interest in the gradient of $\mathscr{E}_m$ with respect to a weight $w_{i;j,k}$ or bias $b_{i;j}$, note that these parameters affect $\mathscr{E}_m$ only through their appearance in Eq. (2). Hence, we obtain an evaluation of all of the elements of the gradient vector in terms of $\delta^m_{i;j}$ through

$$\frac{\partial \mathscr{E}_m}{\partial w_{i:j,k}} = \frac{\partial \mathscr{E}_m}{\partial c^m_{i:j}} \frac{\partial c^m_{i:j}}{\partial w_{i:j,k}} = \delta^m_{i:j} a^m_{i-1:k} \tag{6a}$$

$$\frac{\partial \mathscr{E}_m}{\partial b_{i:j}} = \delta^m_{i:j} \tag{6b}$$

It remains to evaluate $\delta^m_{i;j}$. Note that since $\mathscr{E}_m$ depends upon $c^m_{i;j}$ only through $a^m_{i;j}$,

$$\delta^m_{i:j} = \frac{\partial \mathscr{E}_m}{\partial a^m_{i:j}} \frac{\partial a^m_{i:j}}{\partial c^m_{i:j}} = f_{i:j}(c^m_{i:j}) \frac{\partial \mathscr{E}_m}{\partial a^m_{i:j}}$$

If layer i is hidden, then $\mathscr{E}_m$ depends upon $a^m_{i;j}$ only through its effects on the layer $i+1$ to which it is an input. Hence,

$$\frac{\partial \mathscr{E}_m}{\partial a^m_{i:j}} = \sum_{k=1}^{s_{i+1}} \frac{\partial \mathscr{E}_m}{\partial c^m_{i+1:k}} \frac{\partial c^m_{i+1:k}}{\partial a^m_{i:j}} = \sum_{k=1}^{s_{i+1}} \delta^m_{i+1:k} w_{i+1:k,j}$$

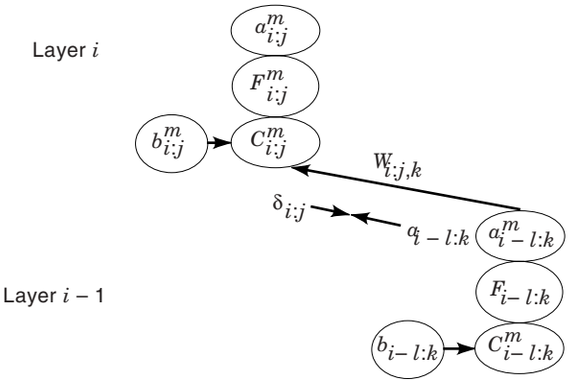

Figure 4. Information flow in backpropagation.

Combining the last two results yields the *backwards recursion*

$$\delta^m_{i:j} = f_{i:j}(c^m_{i:j}) \sum_{k=1}^{s_{i+1}} \delta^m_{i+1:k} w_{i+1:k,j} \tag{7a}$$

for $i < L$. This equation can be rewritten in matrix–vector form using

$$\mathbf{W}_{i+1} = [w_{i+1:k,j}], \ \underline{\delta}^m_i = [\delta^m_{i:j}], \qquad \underline{f}^m_i = [f_{i:j}(c^m_{i:j})]$$
$$\underline{\delta}^m_i = (\underline{\delta}^m_{i+1})^T \mathbf{W}_{i+1\cdot} * \underline{f}^m_i$$

where .* is the Hadamard product (Matlab element-wise multiplication of matrices). The "final" condition, from which we initiate the backwards propagation, is provided by the direct evaluation of

$$\delta^m_{L:1} = f_{L:1}(c^m_{L:1})(a^m_{L:1} - t_m) \tag{7b}$$

Thus the evaluation of the gradient, as illustrated in Fig. 4, is accomplished by:

1. A forward pass of the training data through the network to determine the node outputs $a^m_{i;j}$ and inputs $c^m_{i;j}$
2. A backward pass through the network to determine the $\delta^m_{i;j}$ through Eqs. (7a) and (7b)
3. Combining results to determine the gradient through Eqs. (6a) and (6b)

DESCENT ALGORITHMS

Overview and Startup Issues

The backpropagation algorithm (BPA), in common usage, refers to a descent algorithm that iteratively selects a sequence of parameter vectors $\{\underline{w}_k,\ k = 1:T\}$, for a moderate value of running time T, with the goal of having $\{\mathscr{E}_{\mathscr{T}}(\underline{w}_k) = \mathscr{E}_k\}$ converge to a small neighborhood of a good local minimum rather than to the global minimum

$$\mathscr{E}^*_{\mathscr{T}} = \min_{\underline{w} \in \mathscr{W}} \mathscr{E}_{\mathscr{T}}(\underline{w})$$

Issues that need to be addressed are:

1. Initialization of the algorithm
2. Choice of online (stochastic) versus batch processing

3. Recursive algorithm to search for an error surface minimum
4. Selection of parameters of the algorithm
5. Rules for terminating the algorithmic search
6. Convergence behavior (e.g., local versus global minima, rates of convergence)

The search algorithm is usually **initialized** with a choice $\underline{w}_0$ of parameter vector that is selected at random to have moderate or small values. The random choice is made to prevent inadvertent symmetries in the initial choice from being locked into all of the iterations. Moderate weight values are selected to avoid saturating initially the node nonlinearities; gradients are very small when S-shaped nodes are saturated and convergence will be slow. It has been argued in (64) that the performance of steepest descent for neural networks is very sensitive to the choice of $\underline{w}_0$. In practice, one often trains several times, starting from different initial conditions. One can then select the solution having the smaller minimum or make use of a combination of all the solutions found (21).

The descent algorithm can be developed either in a batch mode or in an online/stochastic mode. In the batch mode we attempt the $(k+1)$st step of the iteration to reduce the total error over the whole training set, $\mathscr{E}_{\mathscr{T}}(\underline{w}_k)$, to a lower value $\mathscr{E}_{\mathscr{T}}(\underline{w}_{k+1})$. In the online mode we attempt the $(k+1)$st step of the iteration to reduce a selected component $\mathscr{E}_{m_k+1}$, the error in the response to excitation $\underline{x}_{m_k+1}$, of the total error. Over the course of the set of iterations, all components will be selected, usually many times. Each version has its proponents. To achieve true steepest descent on $\mathscr{E}_{\mathscr{T}}(\underline{w})$ we must do the batch update in which the search direction is evaluated in terms of all training set elements. In practice, the most common variant of BPA is online and adjusts the parameters after the presentation of each training set sample. The operation of the online search is more stochastic than that of the batch search since directions depend upon the choice of training set term. The online mode replaces the large step size taken by the batch process (a sum over online mode type steps for each training sample) by a sequence of smaller step sizes in which you continually update the weight vectors as you iterate. This mode makes it less likely that you will degrade performance by a significant erroneous step. There is a belief (e.g., see Ref. 64a, p. 157) that this enables the algorithm to find better local minima through a more random exploration of the parameter space $\mathscr{W}$.

Iterative Descent Algorithms

We now enumerate all network parameters (link weights and biases) in a vector $\underline{w} \in \mathscr{W} \subset \mathbb{R}^p$. The basic iterative recursion, common to all of the training methods in widespread use today, determines a new parameter vector $\underline{w}_{k+1}$ in terms of the present vector $\underline{w}_k$ through a search direction $\underline{d}_k$ and a scalar learning rate or step size α_k:

$$\underline{w}_{k+1} = \underline{w}_k + \alpha_k \underline{d}_k \tag{8}$$

Typically, descent algorithms are Markovian in that one can define a state and their future state depends only upon their present state and not upon the succession of past states that led up to the present. In the case of basic steepest descent, this state is simply the current value of the parameter and gradient. In the variation on steepest descent using momentum smoothing, the state depends upon the current parameter value and gradient and the most recent past parameter value. Each of the algorithms in current use determine the next search point by looking locally at the error surface.

We can explore the basic properties of descent algorithms by considering the following first-order approximation [i.e., $f(x) - f(x_0) \approx f'(x_0)(x - x_0)$] to successive values of the objective/error function:

$$\mathscr{E}_{k+1} - \mathscr{E}_k \approx \underline{g}(\underline{w}_k)^T(\underline{w}_{k+1} - \underline{w}_k) \tag{9}$$

If we wish our iterative algorithm to yield a steady descent, then we must reduce the error at each stage. For increments $\underline{w}_{k+1} - \underline{w}_k$ that are not so large that our first-order Taylor's series approximation of Eq. (9) is invalid, we see that we must have

$$\begin{aligned} \underline{g}(\underline{w}_k)^T(\underline{w}_{k+1} - \underline{w}_k) &= \underline{g}(\underline{w}_k)^T(\alpha_k \underline{d}_k) \\ &= \alpha_k \underline{g}_k^T \underline{d}_k < 0 \quad \text{(descent condition)} \end{aligned} \tag{10}$$

One way to satisfy Eq. (10) is to have

$$\alpha_k > 0, \qquad \underline{d}_k = -\underline{g}_k \tag{11}$$

The particular choice of descent direction of Eq. (11) is the basis of steepest descent algorithms. Other choices of descent direction are made in conjugate gradient methods (59).

An "optimal" choice α_k^* for the learning rate α_k for a given choice of descent direction $\underline{d}_k$ is the one that minimizes $\mathscr{E}_{k+1}$:

$$\alpha_k^* = \operatorname{argmin}_\alpha \mathscr{E}_{\mathscr{T}}(\underline{w}_k + \alpha \underline{d}_k)$$

This choice is truly optimal if we are at the final stage of iteration. It is easily verified that for the optimal learning rate we must satisfy the orthogonality condition

$$\underline{g}_{k+1}^T \underline{d}_k = 0 \tag{12}$$

The gradient of the error at the end of the iteration step is orthogonal to the search direction along which we have changed the parameter vector. Hence, in the case of steepest descent [Eq. (11)], successive gradients are orthogonal to each other. When the error function is not specified analytically, then its minimization along $\underline{d}_k$ is accomplished through a numerical line search for α_k^*.

Further analysis of the descent condition can be carried out if one makes the customary assumption that $\mathscr{E}_{\mathscr{T}}$ is quadratic with a representation

$$\mathscr{E}_{\mathscr{T}}(\underline{w}) = \mathscr{E}_{\mathscr{T}}(\underline{w}_0) + \frac{1}{2}(\underline{w} - \underline{w}_0)^T \mathbf{H} (\underline{w} - \underline{w}_0) \tag{12a}$$

in terms of the **Hessian** matrix $\mathbf{H}$ of second derivatives of the error with respect to the components of w_0; $\mathbf{H}$ must be positive definite if $\mathscr{E}_{\mathscr{T}}$ is to have a unique minimum. The optimality condition for the learning rate α_k derived from the orthogonality condition [Eq. (12)] becomes

$$\alpha_k^* = \frac{-\underline{d}_k^T \underline{g}_k}{\underline{d}_k^T \mathbf{H} \underline{d}_k} \tag{13}$$

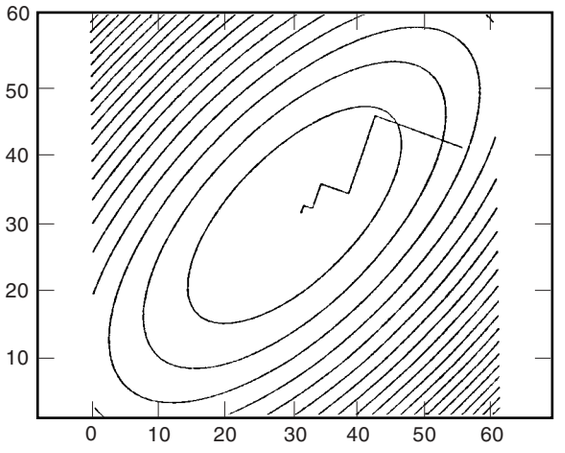

Figure 5. Optimal steepest descent on quadratic surface.

In the further case of steepest descent, Eq. (13) becomes

$$\alpha_k^* = \frac{\underline{g}_k^T \underline{g}_k}{\underline{g}_k^T \mathbf{H} \underline{g}_k} \tag{14}$$

One can think of α_k^* as the reciprocal of an expected value of the eigenvalues of the Hessian $\mathbf{H}$ with probabilities determined by the squares of the coefficients of the gradient vector $\underline{g}_k$ expanded in terms of the eigenvectors of the Hessian. The performance of this choice of learning rate is illustrated in Fig. 5.

Discussion of steepest descent, along with insights into its limitations, is available from Luenberger (58, Chapter 7) and Battiti (56). Steepest descent, even in the context of a truly quadratic error surface and with line search, suffers from an excess of greed. The successive directions do not generally support each other in that after two steps, say, the gradient is usually no longer orthogonal to the direction taken in the first step (e.g., see the contour plot of training trajectory in Fig. 5).

Choice of Constant Learning Rate α

In the basic descent algorithm we follow the above process with the major exception that the step size is held at a constant value α. The simplicity of this approach is belied by the need to select carefully the learning rate. If the fixed step size is too large, then we leave ourselves open to overshooting the line search minimum, we may engage in oscillatory or divergent behavior, and we lose guarantees of monotone reduction of the error function $\mathscr{E}_{\mathscr{T}}$. For large enough α the algorithm will diverge. If the step size is too small, then we may need a very large number of iterations T before we achieve a sufficiently small value of the error function. To proceed further we assume the quadratic case given by Eq. (12a) and let $\{\lambda_j\}$ denote the eigenvalues of the Hessian. It can be shown [e.g., Fine (13, Chapter 5)] that convergence of $\underline{w}_{k+l}$ to the local minimum $\underline{w}^*$ requires, for arbitrary $\underline{w}_k$, that

$$\max_j |1 - \alpha\lambda_j| < 1 \quad \text{or} \quad 0 < \alpha < \frac{2}{\max_j \lambda_j}$$

If α exceeds this upper bound, then $\underline{w}_{k+l}$ must diverge in magnitude. We illustrate these results with plots of the steepest descent trajectory calculated for 25 iterations on a quadratic surface in two dimensions with eigenvalues of 1, 5. Hence, the bound on convergence for α is 0.4. In the next two figures we present four plots with α taking on the values 0.02, 0.1, 0.35, and 0.45. In Fig. 6 we see that a very small learning rate does not allow us to reach the minimum in the allotted training time, whereas a moderately small value enables a smooth approach to the minimum. In Fig. 7 we see that a large value of learning rate enables us to converge to the minimum in an erratic fashion. However, a too large value of learning rate leads to the predicted divergence. It is clear that a useable fixed choice of learning rate requires experimentation with short trial runs of the training algorithm applied to the specific problem at hand. There are a variety of alternatives to the choice of a constant learning rate including the use of adaptive momentum ideas [e.g., Battiti (65)], practical implementations of the optimal learning rate give by Eq. 8.2.7 that rely upon finite difference estimates of derivatives and efficient determination of $\mathbf{H}\underline{g}_k$ and the use of learning-rate schedules [e.g., Darken and Moody (66)]. While the use of constant learning rates was the rule in the recent history of neural network training, state-of-the-art usage would dictate the use of variable learning rates calculated by one of the methods cited.

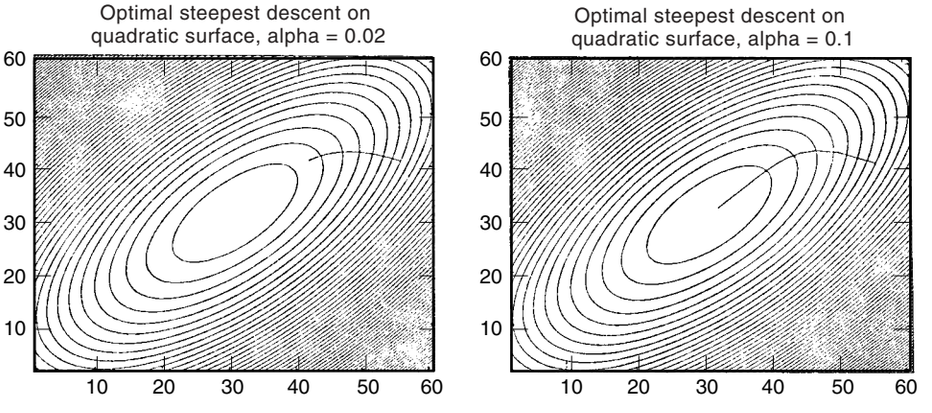

Figure 6. Descent behavior with small learning rate.

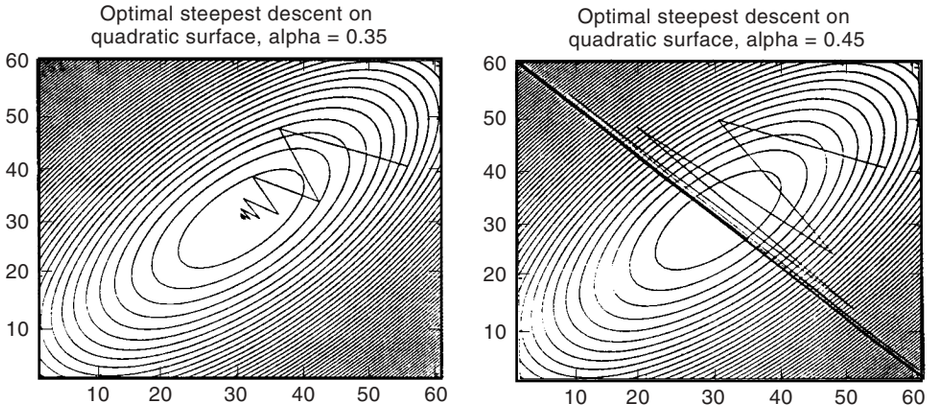

Figure 7. Descent behavior with large learning rate.

Search Termination

Finally, we need to determine when to terminate the search for a minimum of $\mathscr{E}$. Five commonly relied upon stopping conditions to establish termination are:

1. Preassigned upper bound (stopping time) T to the number of iterations
2. Achievement of a preassigned satisfactory value $\mathscr{E}_{final}$ of $\mathscr{E}$
3. Successive changes in parameter values fall below a preassigned threshold
4. The magnitude $\|g\|$ of the current gradient is small, $\|g\| < \epsilon$
5. Increasing estimate (e.g., by "cross-validation" or an independent test set) of generalization error

Several of these conditions are often employed simultaneously. Computational limits generally impose a running time bound T. It may not be clear what a reasonable $\mathscr{E}_{final}$ is unless prior experimentation has provided some indication of what is achievable and the problem is understood well enough that acceptable performance can be identified.

Items 3 and 4 are attempts to judge when convergence is near. In real applications of some complexity, steepest descent algorithms cannot be expected to converge to a global minimum. There can be plateaus in the error surface that eventually lead to good minima.

In the neural network community, frequent reference is made to cross-validation of estimates of generalization error, although this usually turns out to mean the use of an independent test set [e.g., see Kearns (63)]. A validation error $\mathscr{E}_v$ is computed, say, on the basis of a validation or test set $\mathscr{D}_m$ that is independent of the training set $\mathscr{T}$. $\mathscr{E}_v(\underline{w})$ is determined by running the network with parameters $\underline{w}$ on $\mathscr{D}_m$ and evaluating the sum-squared error incurred in fitting $\mathscr{D}_m$. This calculation is repeated as we progress through the sequence $\underline{w}_k$ of parameter values returned by our iterative training algorithm. Training halts when $\mathscr{E}_v(\underline{w}_k)$ reaches its first minimum. Qualitative behavior of the validation error $\mathscr{E}_v$ and the training set error $\mathscr{E}_{\mathscr{T}}$ is shown in Fig. 8. The objective is to guard against *overtraining,* a condition in which the network overfits the training set and fails to generalize well. Target variables usually contain noise as well as signal—there is usually only a stochastic relationship between feature vector $\underline{x}$ and target t, with repetitions of the same feature vector often corresponding to different target values. Fitting too closely to the training set means fitting to the noise as well, and thereby doing less well on new inputs having noise that is independent of that in the training set.

TRENDS AND OPEN PROBLEMS

Progress continues to be made on a number of issues in the design and utilization of neural networks, but several important issues seem to us still in need of development before we can place confidence in their resolution. Of the three issues noted below, the most reliable results are in the use of Hessian-based training methods.

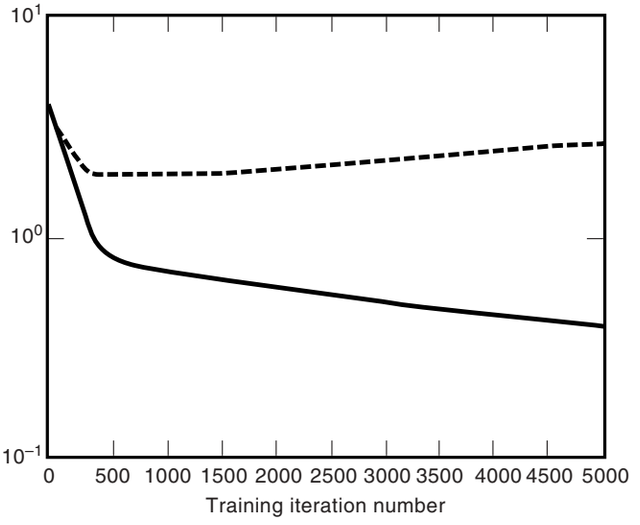

Figure 8. Training and validation errors.

Use of Hessians and Second-Order Methods

The unabated growth in computing power made it possible to train neural networks with backpropagation and steepest descent methods in the mid-1980s. What was once prohibitively expensive computation is now either possible or on the near horizon although we do not foresee having enough computational power to brute force exhaustive searches for the best network architecture and specification of parameters. The most powerful nonlinear optimization procedures (e.g., Newton's method) rely not only on the gradient $\mathbf{G} = \nabla \mathscr{E}_{\mathscr{T}}$ of the error function $\mathscr{E}_{\mathscr{T}}$ calculated by backpropagation, but also on a matrix $\mathbf{H}$ of second derivatives known as the Hessian and given by

$$\mathbf{H} = [H_{ij}], \qquad H_{ij} = \frac{\partial^2 \mathscr{E}_{\mathscr{T}}}{\partial w_i \partial w_j}$$

If the network has p parameters (weights), then $\mathbf{H}$ is a $p \times p$ symmetric matrix and has $p(p + 1)/2$ entries to be calculated. In typical networks, s can be in the hundreds or thousands, yielding possibly millions of Hessian entries. One then faces the burden of (a) updating these many entries as the iterative algorithm proceeds and (b) storing them. A useful discussion of backpropagation-based calculational methods is provided by Bishop (12, Section 4.10). Until recently the primary approach has been to approximate the Hessian by calculating only the diagonal entries. More complex approximations have been used in such so-called second-order optimization methods as the BFGS version of quasi-Newton and the Levenberg-Marquardt algorithms [e.g., see Battiti (56), Buntine and Weigend (67), and Press et al. (61)]. It seems a safe prediction that in the future, neural network training algorithms will rely substantially upon the Hessian in second-order nonlinear optimization algorithms and that for moderate-sized networks this Hessian will be computed fully.

Learning and Generalization Behavior

We know from the section entitled "The Representational Power of a Single-Hidden-Layer Network" that a sufficiently complex network can approximate arbitrarily closely to a given reasonable partially specified function or training set $\mathscr{T}$. What prevents us from attempting arbitrarily close approximations by use of arbitrarily large/complex networks is the desire for good performance on $(\underline{x}, t) \notin \mathscr{T}$. Typically, this issue is formalized by assuming that there is a (unknown to us) probability measure P such that the elements $(\underline{x}_i, t_i)$ of $\mathscr{T}$ are selected independently and identically distributed (i.i.d.) as P and $(\underline{x}, t)$ is also selected by P and independent of $\mathscr{T}$. It is this sense in which $(\underline{x}, t)$ is like the other elements of $\mathscr{T}$. In pattern classification applications the target variable t is discrete and ranges over the finite set of labels of pattern classes. In such a setting it is common to use error probability $P(\eta(\underline{x}, \underline{w}) \neq t)$ as a measure of network η performance. In forecasting, estimation, and control settings, the target variable is typically real-valued and an appropriate error measure is that of mean-squared error $E(\eta(\underline{x}, \underline{w}) - t)^2$.

Analysis of learning and generalization behavior is an unsettled but evolving area. The issues are not particular to neural networks but are rather endemic in statistics and have been long-considered in pattern classification applications [e.g., McLachlan (68)]. An accessible introduction can be found in Ripley (69), Section 2.7, a comprehensive discussion in Devroye et al. (70). One line of development starts from the well-known fact that the observed training error $\mathscr{E}_{\mathscr{T}}(\underline{w})/n$ evaluated at $\underline{w}$ chosen near a local minimum is a biased estimator of $E(\eta(\underline{x}, \underline{w}) - t)^2$ and tends to be too small. An unbiased estimator can be obtained if one has reserved an independent test or validation set $\mathscr{V}$ that is distributed as $\mathscr{T}$ but not itself used in training. For such a set $\mathscr{E}_{\mathscr{V}}$ will be unbiased for true $E(\eta(\underline{x}, \underline{w}) - t)^2$. When the training data are too few to reserve an independent validation set, then methods of cross-validation [Stone in (71,72), Kearns (73)] may be applicable. The bootstrap [e.g., see Efron and Tibshirani (74)] is of doubtful applicability given the presence of multiple minima ensuring that different bootstrap samples are unlikely to converge to the vicinity of the same minimum. Furthermore, both of these methods are computationally very expensive to implement. In another direction, strong efforts to understand the interplay between generalization error, the amount of training data, and the complexity of the model (neural network) being fitted include the work of Barron (50) the concept of Vapnik-Chervonenkis dimension pioneered by Vladimir Vapnik (52), and the recent exploitation of the modification known as the 'fat-shattering dimension' [e.g., Bartlett (75)].

Architecture Selection and Bayesian Methods

One of the thorniest problems facing users of neural networks is that of architecture selection—the selection of node functions [see Mhaskar and Micchelli (76)], numbers of layers, and widths of the layers. While we have seen from the section entitled "The Representational Power of a Single-Hidden-Layer Network" that almost any nonpolynomial node function in a sufficiently wide ($s_1 \gg 1$) single-hidden-layer network will approximate arbitrarily closely to many functions (e.g., continuous or integrable) in an appropriate approximation metric (e.g., sup-norm or L^p-norm, respectively), such an approximation may neither be an efficient use of network hardware, well-suited to sup-norm approximation of piecewise continuous or piecewise constant functions, nor need it generalize well to new data of the same type as that in the training set $\mathscr{T}$. At present, architecture selection is most commonly addressed by hit-and-miss numerical experimentation. A variety of architectures are selected, they are trained on a portion of the training set, and their performance is evaluated on the remaining unused portion of the training set. This process is computationally expensive and may be inapplicable when n, the size of $\mathscr{T}$, is not large (e.g., $n = O(100)$).

The most systematic approach to architecture selection is that based upon the Bayesian methodology. An exposition of this methodology is available from West and Harrison (77) and, in the context of neural networks, from Bishop (12, Chapter 10) and MacKay (78); critical remarks can be found in Fine (13, Chapter 7). In brief, a prior probability distribution is assumed over all architectures and parameter assignments, and this prior distribution is converted to a posterior distribution through a likelihood function incorporating $\mathscr{T}$. The posterior then allows a rational selection of network architecture, particularly when it is sharply peaked about a single network specification.

Other methods of architecture selection are somewhat ad hoc and include methods (e.g., "optimal brain surgery") to prune a network that is chosen initially to be larger than ex-

pected to be needed [e.g., Hassibi et al. (79)] and methods to grow a network until an appropriate level of complexity has been reached [e.g., Fahlman (80) and Gallant (81)] and reliance upon complexity measures as in regularization and the use of minimum description length [e.g., Rissanen (82,83)].

BIBLIOGRAPHY

1. P. Churchland, *Neurophilosophy: Toward a Unified Science of the Mind / Brain,* Cambridge MA: MIT Press, 1986.
2. J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, *Proc. Nat. Acad. Sci.,* **79**: 2554–2558, 1982. Also in Ref. 4.
3. J. Cowan, Neural networks: The early days, in D. Touretzky (ed.), *Advances in Neural Information Processing Systems 2,* San Mateo CA: Morgan Kaufmann, 1990, pp. 828–842.
4. J. Anderson and E. Rosenfeld (eds.), *Neurocomputing: Foundations of Research,* Cambridge MA: MIT Press, 1988.
5. F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms,* Washington, DC: Spartan Books, 1961.
6. F. Rosenblatt, *Psychol. Rev.,* **65**: 386–408, 1958.
7. J. Hopfield, Neurons with graded response have collective computational properties like those of two-state neurons, *Proc. Nat. Acad. Sci.,* **81**: 3088–3092. Also in Ref. 4.
8. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Learning internal representations by error propagation, *Parallel Distributed Processing,* Cambridge, MA. In D. E. Rumelhart and J. L. McClelland (eds.), MIT Press. Also in Ref. 4.
9. V. Roychowdhury, A. Orlitsky, and K.-Y. Siu, Lower bounds on threshold and related circuits via communication complexity, *IEEE Trans. Inf. Theory,* **40**: 467–474, 1994.
10. K.-Y. Siu, V. Roychowdhury, and T. Kailath, *Discrete Neural Computation: A Theoretical Foundation,* Englewood Cliffs, NJ: Prentice Hall, 1995.
11. S. Muroga, *Threshold Logic and Its Applications,* New York: Wiley, 1971.
12. C. Bishop, *Neural Networks for Pattern Recognition,* Oxford: Clarendon Press, 1995.
13. T. L. Fine, *Feedforward Artificial Neural Networks,* New York: Springer-Verlag, 1998.
14. M. Hassoun, *Fundamentals of Artificial Neural Networks,* Cambridge, MA: MIT Press, 1995.
15. S. Haykin, *Neural Networks,* New York: Macmillan, 1994.
16. J. Hertz, A. Krogh, and R. Palmer, *Introduction to the Theory of Neural Computation,* Reading, MA: Addison-Wesley, 1991.
17. B. Ripley, *Pattern Recognition and Neural Networks,* Cambridge: Cambridge University Press, 1996.
18. *IEEE Transactions on Neural Networks,* Piscataway, NJ: IEEE Press.
19. *Neural Computation,* Cambridge, MA: MIT Press.
20. *Neural Networks,* publication of the International Neural Network Society (INNS).
21. *Combining Artificial Neural Nets: Ensemble Approaches,* special issue of *Connection Science,* **8**: December 1996.
22. *Advances in Neural Information Processing Systems,* an annual series of carefully reviewed conference proceeding volumes, with volumes 1–7 published by Morgan Kaufmann Publishers and subsequent volumes published by MIT Press.
23. M. Stevenson, R. Winter, and B. Widrow, Sensitivity of feedforward neural networks to weight errors, *IEEE Trans. Neural Networks,* **1**: 71–80, 1990.
24. P. Kerlirzin and F. Vallet, Robustness in multilayer perceptrons, *Neural Comp.,* **5**: 473–482, 1993.
25. A. Minai and R. Williams, Perturbation response in feedforward networks, *Neural Networks,* **7**: 783–796, 1994.
26. E. Sanchez-Sinencio and R. Newcomb, eds., Special issue on neural network hardware, *IEEE Trans. Neural Networks,* **3**: 1992.
27. C. Mead, *Analog VLSI and Neural Systems,* Reading, MA: Addison-Wesley, 1989.
28. T. Shibata et al., Neuron-MOS temporal winner search hardware for fully-parallel data processing. In D. Touretzky, M. Mozer, and M. Hasselmo (eds.), *Advances in Neural Information Processing Systems 8,* Cambridge, MA: MIT Press, 1996, pp. 685–691.
29. J. Platt and T. Allen, A neural network classifier for the I1000 OCR chip. In D. Touretzky, M. Mozer, and M. Hasselmo (eds.), *Advances in Neural Information Processing Systems 8,* Cambridge, MA: MIT Press, 1996, pp. 938–944.
30. S. Decatur, Application of neural networks to terrain classification, *Proc. IJCNN,* **I**: I283–I288.
31. H. Rowley, S. Baluja, and T. Kanade, Human face detection in visual scenes. In D. Touretzky, M. Mozer, and M. Hasselmo (eds.), *Advances in Neural Information Processing Systems 8,* Cambridge, MA: MIT Press, 1996, pp. 875–881.
32. S. Lawrence, A. Tsoi, and A. Back, The gamma MLP for speech phoneme recognition. In D. Touretzky, M. Mozer, and M. Hasselmo (eds.), *Advances in Neural Information Processing Systems 8,* Cambridge, MA: MIT Press, 1996, pp. 785–791.
33. G. Zavaliagkos et al., A hybrid neural net system for state-of-the-art continuous speech recognition. In S. Hanson, J. Cowan, C. Giles (eds.), *Advances in Neural Information Processing Systems 5,* San Mateo, CA: Morgan Kaufmann, 1993, pp. 704–711.
34. L. Jackel et al., Neural-net applications in character recognition and document analysis, In B. Yuhas and N. Ansari (eds.), *Neural Networks in Telecommunications,* Norwell, MA: Kluwer, 1994.
35. A. Shustorovich and C. Thrasher, KODAK IMAGELINK OCR Alphanumeric handprint module. In D. Touretzky, M. Mozer, and M. Hasselmo (eds.), *Advances in Neural Information Processing Systems 8,* Cambridge, MA: MIT Press, 1996, pp. 778–784.
36. I. Guyon et al., Design of a neural network character recognizer for a touch terminal, *Pattern Recognition,* **24**: 105–119, 1991.
37. J. L. Yuan and T. L. Fine, Neural network design for small training sets of high dimension, *IEEE Trans. on Neural Networks,* to appear.
38. A. Weigend and N. Gershenfeld (eds.), *Time Series Prediction: Forecasting the Future and Understanding the Past,* Reading, MA: Addison-Wesley, 1994.
39. R. Caruana, *Learning many related tasks at the same time with backpropagation,* in G. Tesauro, D. Touretzky, T. Leen, (eds.), Advances in Neural Information Processing Systems 7, Cambridge, MA: MIT Press, 1995, 657–664.
40. G. Gibson, Exact classification with two-layer neural nets, *J. Comput. Syst. Sci.,* **52**: 349–356, 1996.
41. E. Sontag, Feedback stabilization using two-hidden-layer nets, *IEEE Trans. Neural Networks,* **3**: 981–990, 1992.
42. Y. LeCun and Y. Bengio, *Convolutional networks for images, speech, and time series,* in M. Arbib (ed.), The Handbook of Brain Theory and Neural Networks, Cambridge, MA: MIT Press, 1995, 255–258.
43. A. Albertini, E. Sontag, and V. Maillot, Uniqueness of weights for neural networks. In R. Mammone (ed.), *Artificial Neural Networks for Speech and Vision,* London: Chapman and Hall, pp. 113–125.
44. C. Fefferman, Reconstructing a neural net from its output, *Rev. Mat. Iberoamericana,* **10**: 507–555, 1994.

45. G. Cybenko, Approximations by superpositions of a sigmoidal function, *Mathematics of Control, Signals & Systems,* **2** (4): 303–314. Correction made in op. cit., **5**: 455, 1989.

46. K. Hornik, M. Stinchcombe, and H. White, Multilayer feedforward networks are universal approximators, *Neural Networks,* **2**: 359–366, 1989.

47. M. Leshno et al., Multilayer feedforward networks with a nonpolynomial activation function can approximate any function, *Neural Networks,* **6**: 861–867, 1993.

48. K. Hornik, Approximation capabilities of multilayer feedforward networks, *Neural Networks,* **4**: 251–257, 1991.

49. J. Yukich, M. Stinchcombe, and H. White, Sup-norm approximation bounds for networks through probabilistic methods, *IEEE Trans. Inf. Theory,* **41**: 1021–1027, 1995.

50. A. Barron, Universal approximation bounds for superpositions of a sigmoidal function, *IEEE Trans. on Information Theory,* **39**: 930–945, 1993.

51. L. Jones, A simple lemma on greedy approximation in Hilbert space and convergence rates for projection pursuit regression and neural network training, *The Annals of Statistics,* **20**: 608–613, 1992.

52. V. N. Vapnik, *The Nature of Statistical Learning Theory,* New York: Springer-Verlag, 1995.

53. M. Kearns and U. Vazirani, *An Introduction to Computational Learning Theory,* Cambridge, MA: MIT Press, 1994.

54. P. Auer, M. Herbster, and M. Warmuth, Exponentially many local minima for single neurons. In D. Touretzky, M. Mozer, and M. Hasselmo, (eds.), *Advances in Neural Information Processing Systems 8,* Cambridge, MA: MIT Press, pp. 316–322.

55. S. Hashem, *Optimal Linear Combinations of Neural Networks,* Ph.D. dissertation, Purdue University, W. Lafayette, IN, 1993.

56. R. Battiti, First- and second-order methods for learning: Between steepest descent and Newton's methods, *Neural Computat.,* **4**: 141–166, 1992.

57. R. Fletcher, *Practical Methods of Optimization,* New York: Wiley, 1987.

58. D. Luenberger, *Linear and Nonlinear Programming,* 2nd ed., Reading, MA: Addison-Wesley, 1984.

59. M. Møller, A scaled conjugate gradient algorithm for fast supervised learning, *Neural Networks,* **6**: 525–533, 1993.

60. M. Hagan and M. Menhaj, Training feedforward networks with the Marquardt algorithm, *IEEE Trans. Neural Networks,* **5**: 989–993, 1994.

61. W. Press et al., *Numerical Recipes in C: The Art of Scientific Programming,* 2nd ed., Cambridge, England: Cambridge University Press, 1992.

62. A. Tikhonov and V. Arsenin, *Solutions of Ill-Posed Problems,* Winston & Sons through Wiley, Washington, D.C., 1977.

63. A. Weigend, D. Rumelhart, and B. Huberman, *Generalization by weight-elimination with application to forecasting,* in R. Lippmann, J. Moody, D. Touretzky (eds.) Advances in Neural Information Processing Systems 3, Morgan Kaufmann Pub., 875–882, 1991.

64. J. Kolen and J. Pollack, Back propagation is sensitive to initial conditions. In R. Lippmann, J. Moody, and D. Touretzky (eds.), *Advances in Neural Information Processing Systems 3,* San Mateo, CA: Morgan Kaufmann pp. 860–867.

64a. Y. LeCun, P. Simard, and B. Pearlmutter, *Automatic learning rate maximization by on-line estimation of the Hessian's eigenvectors,* in S. Hanson, J. Cowan, L. Giles (eds.), Advances in Neural Information Processing Systems 5, Morgan Kaufmann Pub., 156–163, 1993 and Y. LeCun and Y. Bengio, *Convolutional networks for images, speech, and time series,* in M. Arbib (ed.), The Handbook of Brain Theory and Neural Networks, Cambridge, MA: MIT Press, 255–258, 1995.

65. B. Pearlmutter, Fast exact multiplication by the Hessian, *Neural Computation,* **6**: 147–160, 1994.

66. C. Darken and J. Moody, *Towards faster stochastic gradient search,* in J. Moody, S. J. Hanson, R. P. Lippmann (eds.), Advances in Neural Information Processing Systems 4, Morgan Kaufmann Pub., 1992, 1009–1016.

67. W. Buntine and A. Weigend, Computing second derivatives in feedforward networks: A review, *IEEE Trans. Neural Networks,* **5**: 480–488.

68. G. McLachlan, *Discriminant Analysis and Statistical Pattern Recognition,* New York: Wiley, 1992.

69. B. Ripley, *Pattern Recognition and Neural Networks,* Cambridge Univ. Press, Cambridge, 1996.

70. L. Devroye, L. Gyorfi, and G. Lugosi, *A Probabilistic Theory of Pattern Recognition,* New York: Springer-Verlag, 1996.

71. M. Stone, Cross-validatory choice and assessment of statistical predictions, *J. R. Stat. Soc.,* **B36**: 111–147, 1974.

72. M. Stone, Asymptotics for and against cross-validation, *Biometrika,* **64**: 29–35, 1977.

73. M. Kearns, A bound on the error of cross validation using the approximation and estimation rates, with consequences for the training-test split. In D. Touretzky, M. Mozer, M. Hasselmo (eds.), *Advances in Neural Information Processing Systems 8,* Cambridge, MA: MIT Press, 1996, pp. 183–189.

74. B. Efron and R. Tibshirani, *An Introduction to the Bootstrap,* London: Chapman and Hall, 1993.

75. P. Bartlett, *The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network,* in M. Mozer, M. Jordan (eds.), Advances in Neural Information Processing Systems 9, Cambridge, MA: MIT Press, to appear.

76. H. Mhaskar and C. Micchelli, How to choose an activation function. In J. Cowan, G. Tesauro, and J. Alspector (eds.), *Advances in Neural Information Processing Systems 6,* San Mateo, CA: Morgan Kaufmann, pp. 319–326.

77. M. West and J. Harrison, *Bayesian Statistical Decision Theory,* New York: Springer-Verlag, 1989.

78. D. MacKay, *Bayesian Methods for Adaptive Models,* Ph.D. dissertation, California Institute of Technology.

79. B. Hassibi et al., Optimal brain surgeon: Extensions and performance comparisons. In J. Cowan, G. Tesauro, and J. Alspector (eds.), *Advances in Neural Information Processing Systems 6,* San Mateo, CA: Morgan Kaufmann, 1994, pp. 263–270.

80. S. Fahlman and C. Lebiere, The cascade-correlation learning architecture. In D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems 2,* San Mateo, CA: Morgan Kaufmann, 1990, pp. 524–532.

81. S. Gallant, *Neural Network Learning and Expert Systems,* Cambridge, MA: MIT Press, 1993.

82. J. Rissanen, Stochastic complexity and modeling, *The Annals of Statistics,* **14**: 1080–1100, 1986.

83. J. Rissanen, Stochastic complexity, *J. Royal Statistical Society,* **49**: 223–239, 1987.

TERRENCE L. FINE
Cornell University

FERRITE DEVICES. See FERRITE PHASE SHIFTERS; MICROWAVE SWITCHES.