Chemometrics and Intelligent Laboratory Systems, 18 (1993) 115–155
Elsevier Science Publishers B.V., Amsterdam

Backpropagation neural networks
A tutorial

Barry J. Wythoff
Inorganic Analytical Research Division, National Institute of Standards and Technology, Gaithersburg, MD 20899 (USA)

(Received 25 March 1992; accepted 27 May 1992)

Abstract

Wythoff, B.J., 1993. Backpropagation neural networks. A tutorial. *Chemometrics and Intelligent Laboratory Systems*, 18: 115–155.

Artificial neural networks have enjoyed explosive growth in the past ten years. An indication of the rate of growth of research in this area is the fact that, although the first research journal devoted exclusively to this subject was just introduced in 1987, there are now at least five refereed neural net research journals. These developments are being taken seriously by the semiconductor industry as well: in addition to a host of products developed by smaller firms, Intel, AT&T Bell Labs, Motorola and Hitachi have all introduced silicon implementations of neural network algorithms. Neural networks have a very broad scope of potential application, including many tasks central to chemical research and development. This tutorial begins with a short history of neural network research, and a review of chemical applications. The bulk, however, is devoted to providing a clear and detailed introduction to the theory behind backpropagation neural networks, along with a discussion of practical issues facing developers.

CONTENTS

1. GENERAL INTRODUCTION AND HISTORY OF ARTIFICIAL NEURAL NETWORKS

As is the case with most new technologies reaching a broad scientific audience for the first time, artificial neural networks are currently the subject of a great deal of excitement. Along with this excitement come unrealistic expectations raised by enthusiastic new converts. We will see in this article that neural networks are not really new, nor completely different from classical methods, and that they will not solve all data processing problems. For many difficult problems which have stymied classical methods, however, neural networks can provide an effective solution. They offer the prospect of a usable solution to problems which cannot even be described analytically.

Artificial neural networks are based on simplified mathematical descriptions of what is known about the physical structure and mechanism of biological cognition and learning. The roots of this research field can be traced back to qualitative ideas first published in the early 1940s by McCulloch and Pitts in their paper entitled 'A logical calculus of the ideas immanent in nervous activity' [1]. Therein, McCulloch and Pitts describe binary neurons with fixed thresholds that integrate inputs received through weighted excitatory and inhibitory synapses, to determine the activation state of the neuron (on or off). Through complex formal proofs, they showed that networks employing such neurons could implement any arithmetic or logical function.

In 1949, Hebb published 'The Organization of Behavior' [2], in which he outlined the first plausible learning rule for modification of the synapse weights between neurons. Hebb's postulate can be stated as: synapses used repeatedly in excitation are reinforced, or strengthened.

The first practical computational model was described by Rosenblatt in a 1958 paper entitled 'The perceptron: a probabilistic model for information storage and organization in the brain' [3]. Rosenblatt described a multi-layered hierarchical structure of neurons with localized and random

connectivity. Each node formed a weighted sum of its inputs and compared them to a fixed threshold to determine if the output would be 1 or −1. His initial learning law was a relatively simple reinforcement rule, which later evolved into a mechanism for supervised learning using error feedback. The perceptron was implemented in the analog hardware of the day by Rosenblatt and coworkers, who used the system to experimentally demonstrate the learning, recall and generalization capabilities that they had postulated. They later proved that the perceptron (a linear hyperplane classifier) would converge to a solution, if one exists, for any binary classification problem, in a finite number of steps.

An explosion of interest in neural networks followed, and many interesting and practical applications of these linear learning machines were produced. Among these were chemical applications of perceptrons to classifying infrared and mass spectral data in the late 1960s and early 1970s [4–7]. Two factors contributed to a marked decline of interest and funding for neural network research in the late 1960s and early 1970s. One was the realization that the hardware available then simply could not support the calculation workload required by larger experiments and applications. The second was the influential book 'Perceptrons' published by Minsky and Papert in 1969 [8]. They rigorously proved that networks employing linear transfer functions (such as the perceptron of that time) are incapable of solving nonlinear classification problems. This limitation had been realized previously by other workers. Such was the reputation of Minsky and Papert, however, that their pessimistic analysis of then current as well as future neural networks considerably diminished interest in the field until the early 1980s. During this interval, a number of dedicated researchers quietly made steady progress in both theory and practice.

During the early 1980s neurocomputing underwent a renaissance; research funding began to rise rapidly as powerful new network models and learning rules were developed. Today, there are myriad varieties of neural network architectures and learning rules for both supervised and unsupervised learning, with many new variants reported every year. Despite its history, this is still a very young field — the research literature is each month rife with new theorems, lemmas, and associated proofs. Even a brief summary of the current state of the field is beyond the scope of this article.

Although the achievements of the pioneers of the '40s and '50s may seem to be eclipsed by those of current researchers, it is important to realize that many of the original ideas laid out during those early days are still in use today, albeit in different forms.

2. BIOLOGICAL COMPARISONS

Artificial neural networks, as their name implies, take their inspiration from biological systems. The efforts of the true biological neural modelers are focused on generating a rigorous mathematical description of biological neural activities. The physical, morphological, and chemical structure of the brain are exceedingly complex, however, and ethics preclude invasive studies on the human cortex. While a great many pieces of information are known, there still remains a vast number of missing fragments to the puzzle which is the mystery of the human brain. Therefore, even the most rigorous of current mathematical models are not thought to be accurate, but to provide simply the fewest conflicts with what is known.

While the work of the biophysicists, neuropsychologists, and biomathematicians has been going on, other scientists, mathematicians, and engineers have become interested in the purely abstract properties of connectionist models and their applications. These scientists (such as analytical chemists) are generally less interested in the faithfulness of an artificial neural network than in any new capabilities which the model may provide as an alternative to classical methods.

Both biological networks and backpropagation (or backprop) networks are parallel machines that use a large number of simple processors with a high degree of connectivity, and process information through relatively discrete events. Both sys-

tems use a distributed form of representation or memory. Biological networks and backprop networks are both adaptive systems that learn by adjusting the 'strength' of the connections between neurons in some manner. Table 1 indicates some of the gross differences between a typical backpropagation network implemented on a digital computer, and nature's human neural network.

3. BACKPROPAGATION NETWORKS

3.1. Introduction

We will focus now on a single neural network model: multilayered, feed-forward, backpropagation networks. This choice is not an arbitrary one: (a) backprop networks are capable of implementing pattern association, pattern classification, data compression, robot control, and function approximation tasks, which together encompass the bulk of possible chemical applications; (b) they have been extensively studied, both theoretically and experimentally; (c) they have been the network architecture of choice in the vast majority of practical applications, including chemical applications.

To date, chemical applications of backpropagation networks have included classification of: sugars from ^{13}C NMR data [9], functional groups from infrared spectra [10,11], infrared peaks from infrared spectra [12], functional groups from mass spectra [13], aromatic substitution reaction products [14], protein secondary structure [15,16], jet fuels from chromatographic data [17], odorants from piezoelectric crystal arrays [18], and alloys from glow-discharge atomic emission spectra [19]. Quantitative applications have included wheat samples from near-IR data [20], pharmaceutical components from UV–VIS data [20,21], quantitative structure–activity studies on drugs [22–24], and pork constituents from near-IR data [25]. Recirculating neural networks have been used for spectral data compression [26], and an algorithm for designing minimal networks has been reported in the chemometric literature [27]. A recent review explored many of these works in greater detail [28]. This list will continue to grow rapidly for the foreseeable future.

3.2. Architecture and theory of the feed-forward process involved in prediction

The explosion of interest in backpropagation networks is due to the work of Rumelhart and McClelland [29]. It has since been shown that other workers independently discovered the backprop learning rule as early as 1974 [30]. Rumelhart and McClelland, however, provided lucid descriptions of the architecture and a clear presentation of the derivation of the training rule, along with elegant and simple example applications in their 1986 manuscript. Their communication skills caused rapid dissemination of the new method to other scientists and engineers.

There are two phases to the operation of backprop nets: the forward propagation of activation,

TABLE 1

Some of the differences between a typical backpropagation network simulated on a digital computer, and the human cortex

	Backpropagation	Human cortex
Number of neurons	10^1–10^4	10^{10}
Connection density	5–100 synapses/neuron	10^2–10^4 synapses/neuron
Uniformity	Homogenous	Very heterogenous
Timing	Synchronous	Asynchronous
'Quanta'	Digital	Analog
Response time (1 neuron)	Nanoseconds	Milliseconds
Topological organization	Usually none	Highly organized
Learning modes	Supervised	Supervised and unsupervised
Mechanisms	Deterministic	Deterministic and stochastic

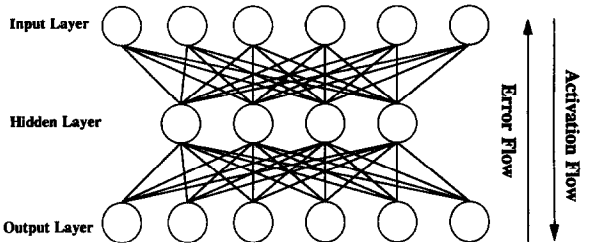


Fig. 1. A fully connected backpropagation network, with the direction of activation and error flow indicated.

which is involved in producing an output result, is described here. The backward propagation of error, which is necessary for 'learning', is described in subsequent sections.

A backpropagation network uses a layered hierarchical architecture of simple neurons (nodes) employing a high degree of connectivity between layers. Only between-layer connections (synapses) are allowed in the 'simple' non-recursive networks described here; no within-layer connections may be used. A schematic diagram of a two layered feed-forward network employing full connectivity between adjacent layers is shown in Fig. 1. It is often overlooked that the backpropagation algorithm does not require complete connection of adjacent layers — restricted connectivity schemes may be employed. Also permitted are synapses which 'skip' one or more layers. The only real restriction is that activation can only flow forward in the network, not backward, laterally, or recursively.

The 'input layer' performs no processing on its inputs, and serves merely to distribute them to the first processing layer. For this reason, it is often not counted in reporting the number of layers in a network architecture, as will be done here. Following the input layer are one or more 'hidden layers', so called because they receive no input from, and produce no output to, the outside world. Finally, the 'output layer' produces the output results of the network for the user. The number of input nodes is fixed by the number of input variables provided for the task, and the number of output nodes is fixed by the number of values which are desired. These would normally correspond to the number of independent and dependent variables involved in the problem, respectively.

Each node in the network receives one or more inputs from the outside world or from preceding layers, and produces a single output value which is broadcast to other node inputs in succeeding layers. The equations which follow are expressed from the viewpoint of a single node, and should be understood to be carried out over the entire network. The first step in calculating the output of a given node is to determine the net input, which is just the dot product of the node's weight vector with its input vector:

$$a = \sum_{i=1}^{n} w_i x_i + \theta \qquad (1)$$

where x_i represent the inputs to the node; w_i represent the weights applied to those inputs; θ is the offset or bias term for the node; and n is the number of synapses for the node.

Next comes the determination of the node output, which is performed by passing the net input of the node through its transfer function.

Although any continuously differentiable (the derivative is defined over the entire function domain) and monotonic (the function is either continually increasing or decreasing over the entire domain) transfer function may be employed in backpropagation networks, those used most often are linear and sigmoidal functions. Linear nodes may be used in conjunction with nonlinear nodes, for scaling the input or output, or to perform feature compression by generating a reduced linear combination of a preceding layer's outputs. A nonlinear transfer function in a multilayered neural network allows it to perform nonlinear functional mappings. There are several forms of equations which may be used to implement a sigmoidal transfer function, including the hyperbolic tangent, but the most often used form is shown below:

$$o = \frac{1.0}{1.0 + e^{-a}} \qquad (2)$$

where o is the node output, and a is the net input, from Eqn. 1.

A one-dimensional sigmoid function was generated by using Eqn. 2, and is shown along with its first derivative in Fig. 2a. This is the type of function that would be produced by a node with one input. The weight on the single input was fixed at 1.0, and the offset at 0.0 for the plots shown in Fig. 2a. For future reference, it is important to note that, with this weight and offset, the net input is equal to the external input, so Fig. 2a also shows the response of the sigmoid with respect to its net input. The effects of varying the offset from −10.0 to 10.0, by steps of 2.0, is shown in Fig. 2b. Varying the offset serves to translate the sigmoid along the x axis. When a positive input weight is used, positive offsets shift the sigmoid center to more negative values, and negative offsets shift it to more positive values. A negative input weight produces the converse behavior.

The effects of varying the weight from 0.25 to 4.0, by factors of 2, is shown in Fig. 2c. The sigmoid is a centrosymmetric function, with the

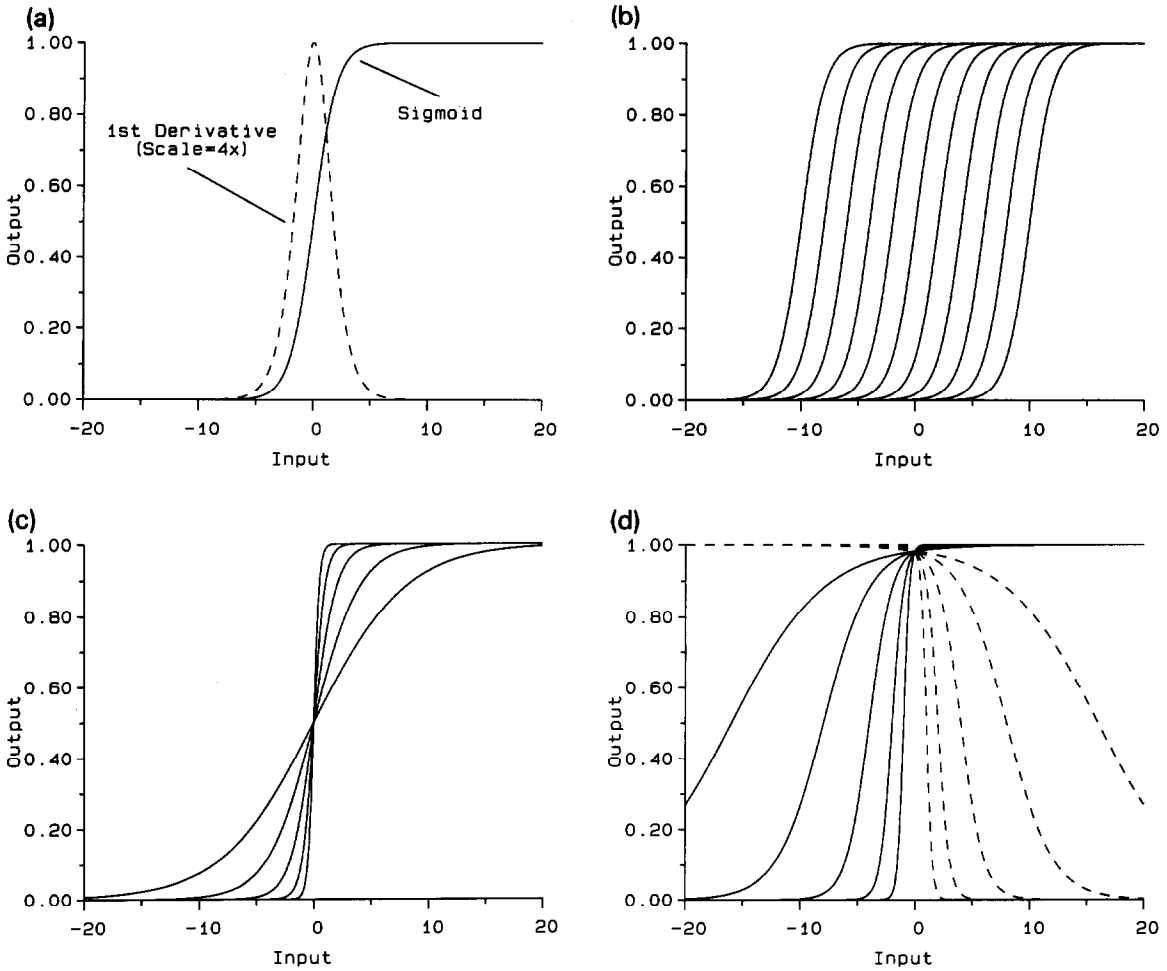


Fig. 2. A one-dimensional sigmoid function as would be formed by a single neuron with one input. (a) The sigmoid and its first derivative; (b) when a positive input weight is used, a positive offset shifts the active region toward values more negative from zero, and a negative offset shifts the center to more positive values; (c) increasing the input weight increases the steepness of the sigmoid response; (d) a positive offset combined with: positive weight values (solid lines) and negative weight values (dashed lines).

center located at output 0.50. It can easily be verified by the reader that this occurs when the net input is zero, and therefore occurs at:

$$\theta = -\sum w_i x_i \quad (3)$$

Increasing the weight magnitude produces a steeper sigmoid function, and in the special case of zero offset, as shown in Fig. 2c, does not change the location of the center (refer also to Eqn. 3). The solid lines in Fig. 2d show the result of varying the weight through the same range as was used in generating Fig. 2c, when an offset of 4.0 is used. Larger weights shift the center to more positive values, as well as increase the slope. Finally, the dashed lines in Fig. 2d show the sigmoid output when an offset of 4.0 is used, and the weights were varied from −0.25 to −4.0, by factors of 2. The effect of reversing the weight polarity while retaining the same offset is to reflect the function output about the line input = 0.0. If both the weight and the offset polarities are reversed, then the response function is reflected about its center.

The adjustment of the network weights during training provides the adaptive fitting capabilities of backprop nets. It is important to note that since the input domain of the sigmoid is bounded during training (a finite range of input magnitudes is presented), the weight adjustments can also serve to 'select' a portion of the full shape of the sigmoid function to be observed over this input domain, by performing the appropriate affine transformation.

Some important characteristics of this sigmoidal function are that it is a centrosymmetric analog threshold logic function, with a smoothly varying output, bounded at 0.0 at $-\infty$, and 1.0 at $+\infty$. The first derivative is very small when the function center is approached from the left (providing resistance to noisy inputs) or from the right (providing stability to the learning process, as explained later). In fact, the output of the sigmoid function is approximately equal to one constant (0.0) over roughly half the domain of the set of real numbers and another constant (1.0) for the complementary set elements, under ordinary conditions. All the significant variation is normally contained in a narrow portion of the input domain, this portion will hereafter be termed the 'active region' of the sigmoid.

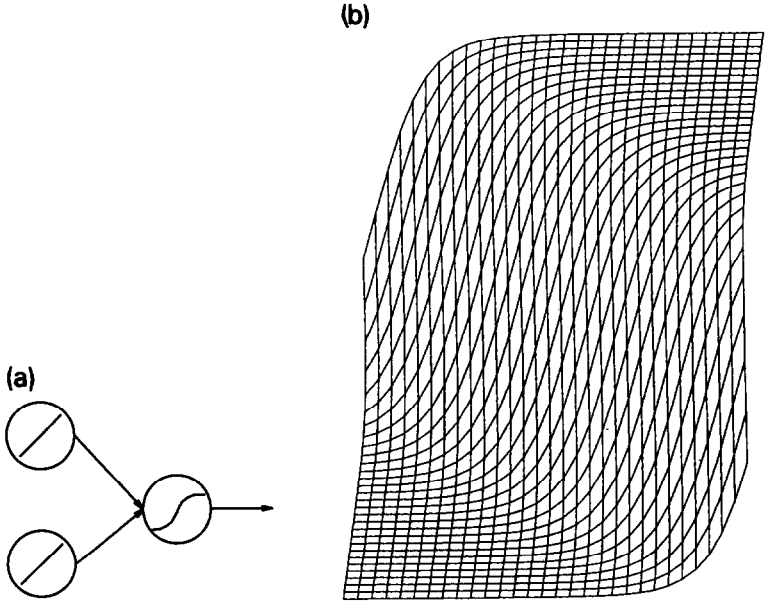


Fig. 3. A two-dimensional sigmoid function, as would be generated by a neuron with two inputs. (a) Schematic of the 'network'; (b) surface plot of the response in the two-dimensional input space; (c) contour plot of the response.

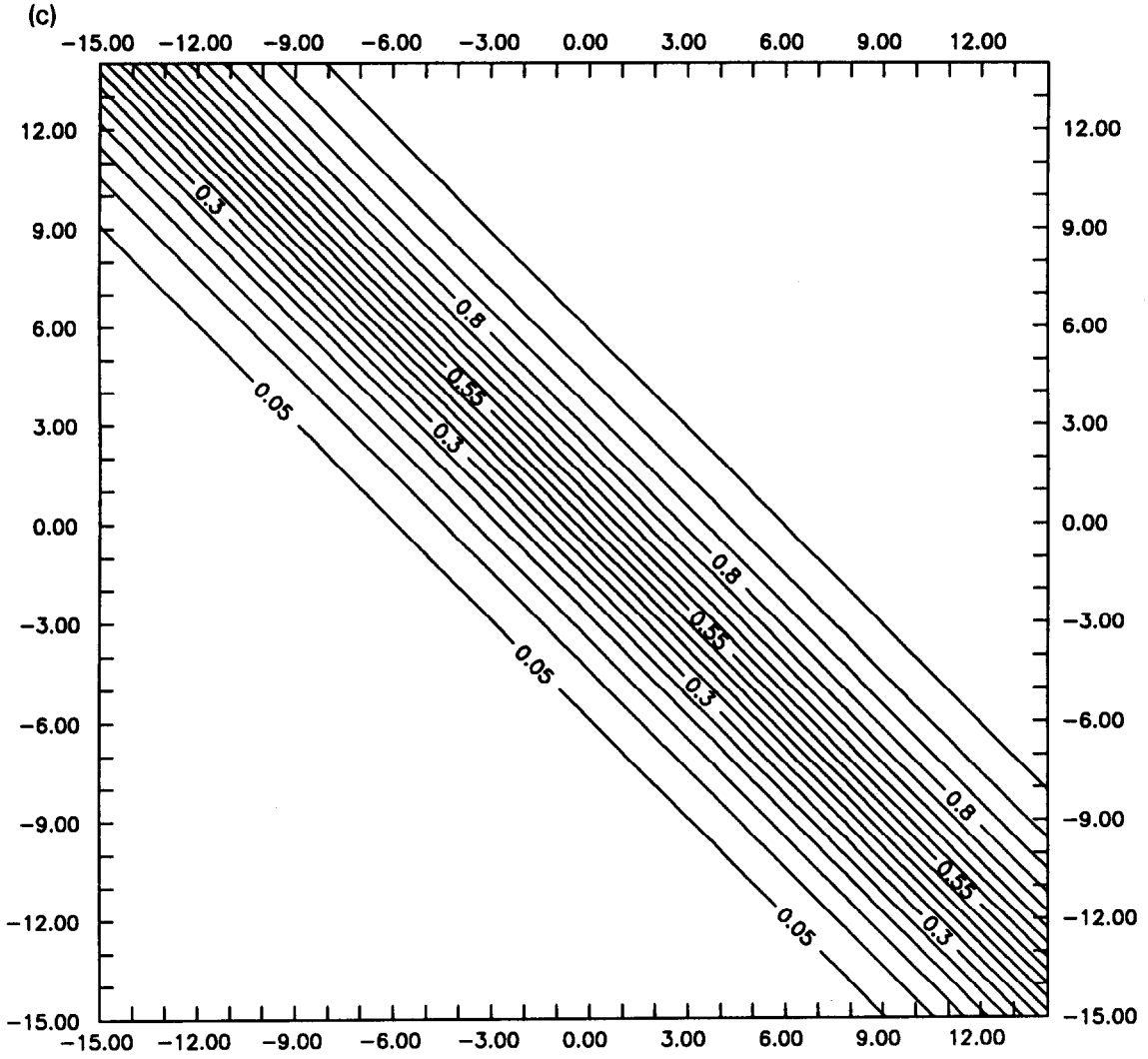


Fig. 3 (continued).

A two-dimensional sigmoid, corresponding to a node receiving two inputs, is shown in Fig. 3. It is helpful to mentally visualize the two individual parts of the activation equation: if x is the input vector, then the calculation of the net input can be symbolically represented as $f(x)$, and the calculation of the output as $g(f(x))$. Calculation of the net input (forming $f(x)$) for a single input (independent variable) constitutes the equation of a line. Passing this dependent value through the transfer function (forming $g(f(x))$) modifies the straight line to give the S-shaped response shown in Fig. 2a. Similarly, the equation of the net input for two independent variables forms the equation of a plane. The sigmoidal transfer function modifies the flat plane to produce the S-shaped plane shown in Fig. 3b. As before, changing the offset serves to translate the active region in the input space. With multidimensional sigmoids, changing the relative values of the weights serves to rotate or orient the active region in the input space, while changing the absolute weight values serves to alter the slope (and also size) of the active region.

In a network employing sigmoidal nodes, each layer of m nodes receiving n inputs can be considered to individually perform a nonlinear projection of the data from an n-dimensional input space to an m-dimensional output space. If m is less than n, then it may be considered that a data reduction is performed by such a layer. The network learns by positioning, orienting, and 'sloping' the active regions of its sigmoids in the input space of each layer, so as to approximate the overall mapping desired.

3.3. Backpropagation networks and supervised learning

Backpropagation nets are used to implement supervised learning tasks, i.e., tasks for which a representative number of example inputs and correct outputs are known. All backprop networks work by learning to approximate a unidirectional mapping from an n-dimensional input space R^n, (where n is the number of input variables) to an m-dimensional output space R^m (where m is the number of output variables). Learning is accomplished by adjusting the weights shown in Eqn. 1, using error feedback from the training examples, so as to bring the network estimates of the correct outputs for the training patterns closer to the true values. The fundamental concept involved in all backpropagation applications is that we are using an ensemble of relatively simple functions (e.g. sigmoids), and combining them in both additive and embedded fashions in order to approximate a complex unknown mapping function.

3.4. Regression and backpropagation learning

The metric used to measure performance on backprop nets is normally (other metrics are sometimes used) the mean squared error (MSE), defined as

$$E = \frac{1}{n} \sum_{i=1}^{n} \left[\sum_{j=1}^{m} (y_{ij} - o_{ij})^2 \right] \tag{4}$$

where y_{ij} represent the correct outputs for a pattern i; o_{ij} represent the network estimates for pattern i; m is the number of output nodes; and n is the number of training patterns. This can readily be seen to be a least squares error criterion just as is used in classical regression analysis. The fundamental difference is that, while classical regression begins with the assignment of an explicit model generated using analytical knowledge of the system under study, backpropagation develops an implicit model. The form of this model is constrained only by the bounds on the set of all functions that the chosen architecture can implement. Therefore, backpropagation learning can be considered to be a generalization of classical regression, a sort of 'super regression'. It is interesting to note, therefore, that the title of Werbos' 1974 Harvard dissertation, which contains the first known published derivation of the backprop learning rule, is entitled 'Beyond Regression' [30].

Just as in classical regression analysis, since the regression performed in backprop learning is a nonlinear form, it must be accomplished iteratively, rather than by direct solution through matrix inversion, as is done for linear regression problems. A form of gradient descent function minimization is implemented in backprop learning to solve the problem of minimizing Eqn. 4. Although it is tempting to propose that we require at least as many training patterns as the total number of weights in the network to solve the system, this is not necessarily so. Such an idea is based loosely on the fact that in order to specify a unique solution to a mapping problem, if it exists, then the number of independent constraints available must be greater than or equal to the number of independent free parameters. One difficulty with this line of reasoning is that training patterns with more than one output value may provide a corresponding number of independent constraints per pattern.

Despite claims by many network proponents that backpropagation networks are superior to conventional methods, this is not always true. Consider a situation where the variance in a data set may be adequately described by a known theoretical model, for example a linear model:

$$r = k_1 x + k_2 + \epsilon \tag{5}$$

where r is the experimental response, k_1 and k_2 are the experimental constants to be fitted, and ϵ represents random error. The deterministic portion of the variation is completely described by this model. The very meaning of the term ϵ is that it cannot be predicted. Therefore, assuming that the ϵ is homogenously distributed over the input variables, the only way to reduce the uncertainty of the predictions of the model is to reduce the average magnitude of ϵ, through better instrumentation or sample manipulation, not to substitute a neural network.

Now consider a data set with n degrees of freedom (d.o.f.), and an underlying deterministic behavior (the analytical description of the system) containing m d.o.f., where $m < n$. The $n - m$ degrees of freedom which remain are due to errors in the data. The pitfall of using a neural network to generate the mapping function, rather than fitting the analytical equation, is that the neural network has no explicit description of the functional model, and so can make no distinction between deterministic and stochastic variation in the data. The network will use all available d.o.f (determined by the architecture) to reduce the MSE for the data. If the network has many more d.o.f than the system (m), this will result (if the training process converges to the global minimum) in overfitting, or 'fitting to the noise' in the data.

It has been observed experimentally for some time that the use of n hidden nodes to learn a mapping on a training set of m datapoints, where $n \geqslant m$, can result in 'grandmothering' behavior, where the network serves as a simple lookup table for the training set. There has been much theoretical effort directed toward establishing bounds on the mapping capabilities of various network architectures and transfer functions. We will informally present here some of the practical consequences of recent theorems published by Sontag [31,32]. The manuscripts cited consider the mapping capabilities of a feedforward network with a single hidden layer, and employing some commonly used transfer functions.

With respect to the lower bounds on classification performance, Sontag has shown that such a network employing n sigmoidal nodes is capable of dividing any arbitrary set of $2n$ points, regardless of the dimensionality, into two arbitrary subsets. Clearly, if the training procedure is successful in locating the global minimum, we will never need to use more than half as many hidden nodes as training patterns to perform any binary classification task. If we have a sufficient number of training patterns to define the true shape of the class boundaries, we can expect to use far fewer to achieve proper generalization. The network limit described above could even resolve training patterns which were 'randomly mixed' in the input space, and we should not expect meaningful data to have this distribution.

Sontag also derived results on the interpolation capabilites of networks employing a single hidden layer of sigmoidal nodes, relevant to quantitative chemical analysis applications. Again, we consider here only the lower limit on such capabilities. Sontag found that a network employing n sigmoidal hidden nodes can approximate to an arbitrary degree of accuracy the response of any $2n - 1$ datapoints, regardless of the dimensionality of the input. Again, this lower limit on performance places upper bounds on the architecture required for a given training set. While a great many questions remain unanswered, results such as these are emerging which are beginning to clearly define the sometimes nebulous nature of neural network research and application issues.

The discussion above also leads to the following generalization: neural networks should only be used when the equations describing the variation in a system are unknown, or cannot be solved. Attempts to do otherwise will produce a solution which can at best match the predictive performance of the analytical solution on unknown inputs, and will probably be worse.

3.5. Generalization, interpolation and extrapolation

A close fit of the neural network model to the training data is not the cardinal issue for the tasks which we seek to solve with them. If all that is desired is to recall a particular output pattern when presented with a particular input pattern, then no model, either explicit or implicit, is

needed for the system. Instead, a simple lookup table is sufficient.

What is desired is to produce a valid estimate of the correct output when a novel input is presented to the trained network. This capability is termed generalization, and can only be measured using test inputs which have not been used to guide the network training process. We can distinguish two types of generalization: interpolation, and extrapolation. Interpolation is measured by using test inputs which lie within the boundaries of the training inputs. Extrapolation is measured by using test inputs which lie outside the boundaries formed by the training inputs.

A neural network with the appropriate architecture, properly trained, will possess good interpolative capabilities. Extrapolation results, however, are often unsatisfactory. The reason for this is simply that the equation which is actually fitted is not the analytical description for the system, and has only been constrained to behave well within the bounds of the training set. It is useful to consider, for example, that in a simple 3:2:1 network employing three input variables, two hidden layer nodes, and one output node, the equation which is fitted is:

$$r = \{1 + \exp[-(w_{211}\exp[-(w_{111}x_1 + w_{112}x_2 + w_{113}x_3 + \theta_{11})] + w_{212} \times \exp[-(w_{121}x_1 + w_{122}x_2 + w_{123}x_3 + \theta_{12})] + \theta_{21})]\}^{-1} \quad (6)$$

where the first subscript on the quantities indicates the row in the network, the second indicates the node, and the third, where used, indicates the synapse; the x_i here are the external inputs to the network; and the w_{ijk} and θ_{ij} are the parameters to be fitted.

With proper adjustment of the weights according to the MSE criterion the network may produce good results over the domain of the training set and for intermediate values. However, the lack of an analytical model becomes a critical issue when trying to extend the empirical model generated to values outside the training domain. With no training data to constrain the model in this region, the network may exhibit pathological behavior on extrapolation. Although extrapolating a valid theoretical model is often risky as well, due to uncertainty in the model parameters or to a limited range of model validity, this is nonetheless an issue on which an analytical solution is preferable.

3.6. The learning equations

In order to minimize the expression of Eqn. 4 by adjusting the weights, we need to have an expression for the partial derivative of the error with respect to each weight:

$$\Delta w_i = k\frac{\partial E}{\partial w_i} \quad (7)$$

We will not derive this expression here — the derivation is very nicely sketched by Rumelhart and McClelland [29], and involves repeated application of the chain rule for partial derivatives, and some clever substitutions. The backprop learning law, also known as the 'generalized delta rule' is given by:

$$\Delta w_j = \eta\delta_i x_j \quad (8)$$

where x_{ij} represents the jth input to the ith node on that pattern presentation; η is the 'stepsize' parameter; and δ_i is the 'delta' term representing the error for the ith node.

The stepsize parameter is a user adjustable parameter which allows some control over the size of the weight changes during training. The delta term is related to calculable quantities for an output node by:

$$\delta_i = (y_i - o_i)[o_i(1 - o_i)] \quad (9)$$

where o_i is the actual output for node i, and y_i is the correct output for node i.

The second term in Eqn. 9 $[o_i(1 - o_i)]$ represents the derivative of the activation function with respect to the net input. This term must be modified appropriately if a transfer function other than Eqn. 2 is to be used.

Although the first term, representing the error portion of Eqn. 8, may be directly specified for an output layer node, it cannot be determined directly for a hidden layer node. Instead, it is

defined recursively in terms of the delta values for the nodes in the layers above it. These delta values are passed back through the synapses, or 'backpropagated', so that the error value for a hidden layer node (i) is taken as a weighted sum of the errors of the nodes in the layers above to which it is connected:

$$\delta_i = [o_i(1 - o_i)] \sum_{j=1}^{n} w_{ij}\delta_j \qquad (10)$$

where w_{ij} is the weight on the output line of node i to node j; δ_j is the δ value for the node j in the layer above which is 'pointed to' by the synapse with weight w_{ij}; and n is the number of synapses for node i.

It can be seen from Eqn. 8 that when using incremental learning (a correction is made after each pattern presentation) the weight vector of a given node must move parallel to the line defined by its input vector on that presentation. This is why weight plots of nodes located in the first hidden layer often show the same features as the input patterns.

Consider a network employing sigmoidal nodes which is being used for classification, with either a separate output node, or even a separate network, for each class. Normally, an output value of 1.0 is requested for class presence, and 0.0 for class absence. In such cases, and where certain nodes in the first hidden layer are associated with a particular output class, the polarities in the features visible in the weights will be opposite for the target class and for competing classes. This is because errors being fed back from the output nodes will have opposite signs for the target class and for competing classes. This can be stated with certainty because the sigmoidal function approaches the values of 0.0 and 1.0 asymptotically, and so the network estimate for class presence must always be low (less than 1.0), and for class absence must always be high (greater than 0.0). Knowledge of this opposed polarity of error feedback, and hence, weight adjustment, allows interpretation of the features observed in weight plots in terms of the uniqueness of characteristic features in the input, for classification problems.

As alluded to above, even when incremental learning is employed, the weight vector plots need not appear like any of the input patterns, however. The final form of the weight vector is determined from the summation of many steps of different magnitudes and signs (corresponding to the δ term of Eqn. 8 at each step). Only each individual step is oriented along the line defined by a given pattern vector. A particular neuron with n weights that experiences n linearly inde-

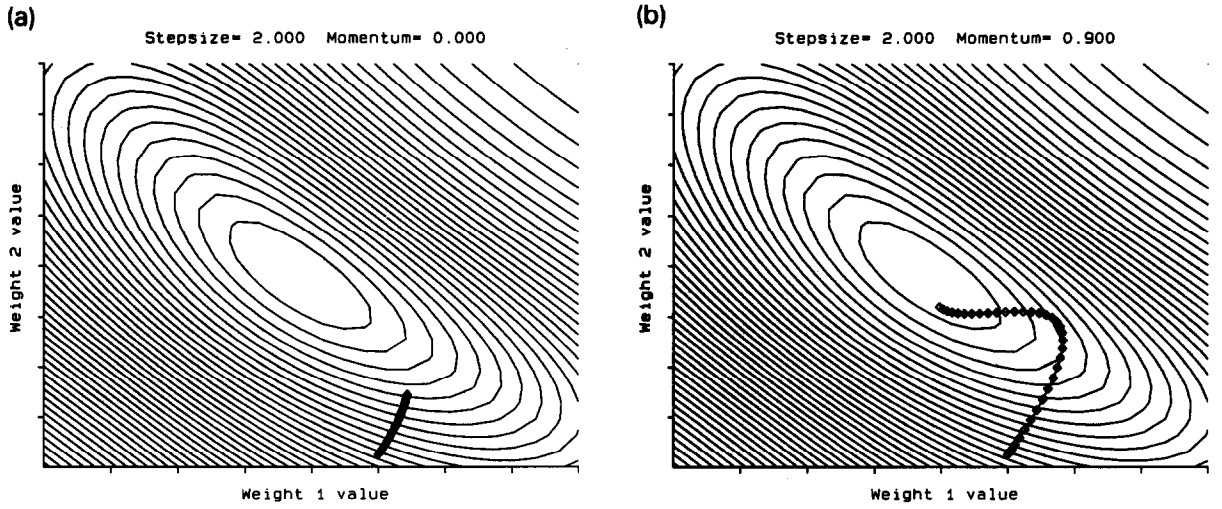


Fig. 4. Training progress on the problem $\cos(x_1 + x_2)$ for a system with two weights. The surface shows the mean squared error as a function of the two weight values. (a) Stepsize = 2.0, momentum = 0.0; (b) stepsize = 2.0, momentum = 0.90.

pendent input patterns during training has access to the entire n-dimensional weight space belonging to it. Fewer than n linearly independent input patterns will allow access to only a subset of the weight space for that neuron, because they cannot completely span its weight space.

3.7. Sampling the training and test sets

In developing a backpropagation solution, it is essential to have a sufficient number of known examples for training and validation. A 'sufficient number' constitutes enough examples to constrain the functional mapping learned by the network into the proper form. Data sets exhibiting more complex functional behavior such as multiple interactions between variables, disjoint and/or overlapping pattern classes, etc. will require more training examples to completely constrain the model, and more testing examples to properly validate the model performance.

The training and test examples can be considered to be drawn by discretely sampling the entire 'world' of possible input/output pairs for the problem. This sampling should be performed randomly, and with the same probability distribution as the world of possible inputs/outputs. If these conditions are met, then as the size of the training set is increased, the functional mapping learned will converge to the behavior which would be observed if the entire problem world were sampled. Similarly, as the size of the test set is increased, then for a given network, the mean squared error measured on the test set will converge to the idealized world value. If it is known that an adequate training set was used, the proper size for a test set can therefore be established by using progressively larger test sets, and looking for convergence of the mean squared error for the test set [33]. Alternatively, to verify adequate training and test sets, we can repeatedly train and test the same network on a data set, dividing the training data into training and test sets of fixed proportion, and randomly choosing the elements of each set. If adequate test and training data are available for the number of degrees of freedom present in the network, then stable MSE values will be observed on both the training and test sets, regardless of the random elements placed in either.

Another important consideration for training set sampling is that the error minimization can be seen from Eqn. 4 to take on any bias present in the training set. Consider, for example, a classification problem for which class A has many more examples in the training data than any other class. The resulting network will likely perform best on discrimination for class A, because the error minimization will be biased toward it, due to its greater frequency of appearance in the training data. One simple way to overcome this sort of behavior is to divide the training data into separate 'bags', one for each class, and to sample from each bag with equal likelihood. This does not make very efficient use of the training data, however.

Curry and Rumelhart [13] interpreted the class frequency bias as the Bayesian class prior probabilites, and used an error weighting system to eliminate that bias. This author has used an analogous weighting system, which will be termed 'fractional complement error weighting' here:

$$E_i = c_i(y_i - o_i)^2 \tag{11}$$

where E_i is the error value for pattern i; c_i is equal to $1-(N_A/N_T)$ if the pattern belongs to class A, and c_i is equal to (N_A/N_T) if the pattern does not belong to class A; N_A is the number of

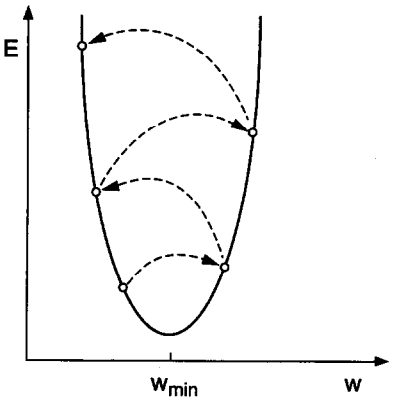


Fig. 5. A conceptual diagram of a divergent trajectory in a one-dimensional quadratic weight error space.

patterns belonging to class A; and N_T is the total number of patterns.

These weights are proportional to those used by Curry and Rumelhart. The latter system employed weights of $1/(N_A/N_T)$ for patterns containing class A, and $1/[1-(N_A/N_T)]$ for patterns not containing class A. It is not entirely clear from the text of their manuscript [13], but it appears that Curry and Rumelhart still normalized the weight error derivative sums by the number of patterns sampled, whether or not error weighting was used. This might explain why they reported that they needed to use a lower value of the stepsize parameter, when "...prior probability weighted errors..." were used, because they found that such weighting was "...causing an uneven descent." In any case, the present author has found a very simple modification to the normalization factor to be useful. The error sum may alternatively be normalized with the sum of the pattern error weights encountered during the batch, just as is done when calculating any conventional weighted mean. Such a scheme places the individual error weights in proper relative

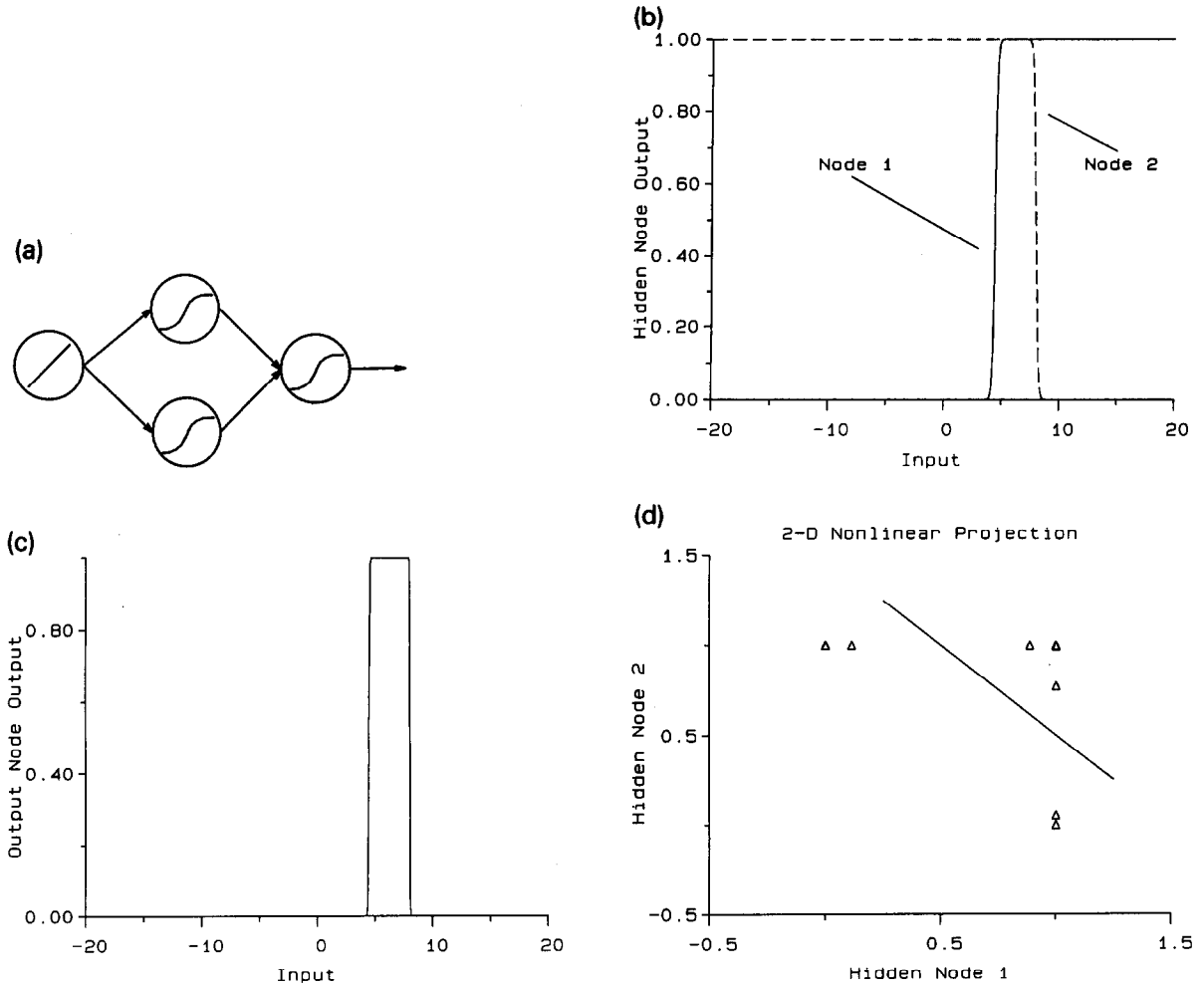


Fig. 6. A 1:2:1 network solution to a nonlinear one-dimensional classification problem. (a) Schematic diagram of the network; (b) response of hidden nodes 1 (solid line) and 2 (dashed line); (c) output node response; (d) response of hidden node 1 versus that of hidden node 2, along with the contour of the sigmoid of the output node at output = 0.50 in the hidden node space.

proportion, while still preserving the 'correct' magnitude for the overall error sum, and hence weight error derivative vector length.

Finally, the training patterns are normally sampled in random, rather than sequential order. When full batch mode learning is used (see below), there are no practical differences between random or sequential pattern sampling. If training steps are made after sampling only a subset of the training patterns, however, then random sampling of the training patterns will decrease the likelihood of becoming trapped in a repetitive cycle of nonproductive steps on the error surface.

3.8. Application of the learning equations

3.8.1. Local and multiple minima

One can consider the training process to be a search for the global minimum on an n-dimensional error surface, for which n is the total number of adjustable weights in the network. With multilayered nonlinear neural networks, local minima may exist on the error surface. In addition, the error surface has multiple 'copies' of the various minima, due to degeneracy arising from symmetry in the network architecture. In order to rationalize this redundancy, consider what would happen to the entire network mapping function if any hidden layer node is swapped with another in the same layer, and their input and output synapses are transferred along with them: there would be no change in the network mapping function. The larger the number of hidden layer nodes, the more 'duplicate' minima that are present, as this is essentially a combinatorial phenomenon.

While the existence of multiple global minima is (generally) good news, the existence of *any* local minima is bad news. Other than exploring the entire error surface, there is no way to guarantee that a minimum which has been located on an arbitrary error surface is the global minimum. This difficulty with local minima is shared with a great many alternative methods, however.

3.8.2. Incremental versus batch mode learning

It can be seen from Eqn. 4 that in order to rigorously calculate the weight error derivatives for the training set, it is necessary to sum the errors over the entire training set, before making

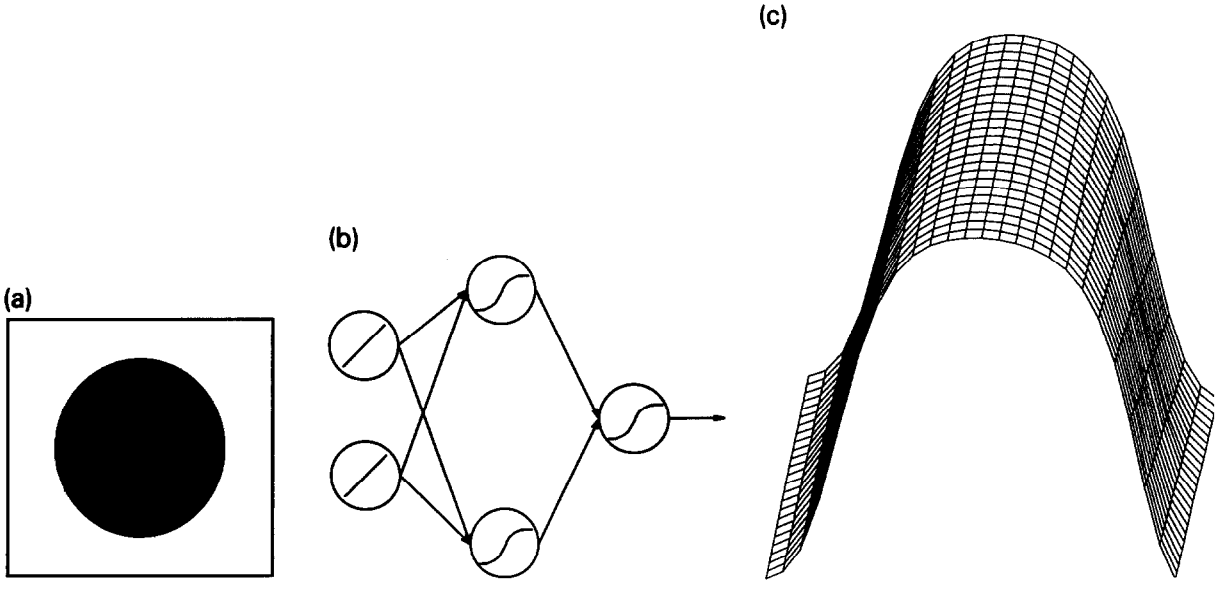


Fig. 7. A 2:2:1 network solution to the circle in a square problem, representing a single class modeling problem. (a) Schematic representation of the problem; (b) schematic diagram of the network architecture; (c) surface plot of the fitted response surface in the two-dimensional input space; (d) contour plot of the fitted response surface.

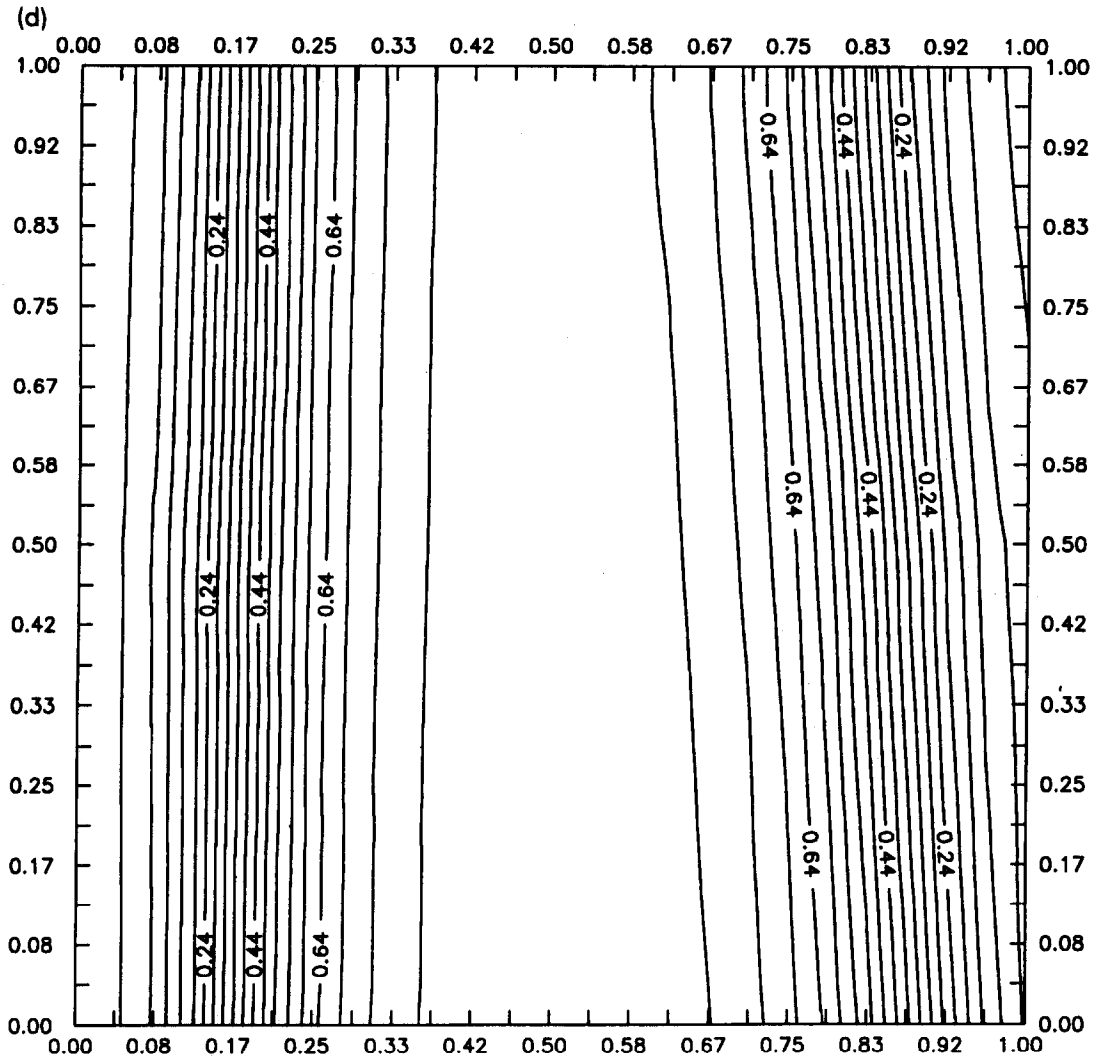


Fig. 7 (continued).

a correction to the weights. This is termed full batch mode learning. At the other extreme, we can use the estimate of the gradient provided by measuring the error on a single pattern, and correct the weights after each pattern presentation. This latter method is termed incremental learning. The coarse estimate of the gradient generated during incremental learning with random pattern selection has the effect of superposing a stochastic component on the search process, which can be beneficial for the reasons described above. In between full batch and incremental learning, one can form improved estimates of the true gradient by examining a random subset of the training patterns before each weight correction. Often, using just a small fraction of the training set provides a sufficient estimate of the gradient, and the more frequent weight updates provide a faster solution than full batch mode learning.

If a form of batch mode learning is used, then the weight error derivative should be divided by the number of patterns used in estimating it, prior to applying Eqn. 8. This practice allows

meaningful comparison of the effects of different learning parameters, even when different numbers of patterns are used to estimate the weight derivatives.

3.8.3. The momentum term

An almost universally used modification to the basic backpropagation learning rule (Eqn. 8) is

$$\Delta w_i(t) = \eta \delta_i x_i + \alpha[\Delta w_i(t-1)] \quad (12)$$

where t is an index denoting the discrete time steps involved in learning, and α is the momentum constant.

The right-hand term in Eqn. 12, called the 'momentum term', causes a fraction of the previous step to be added to the current step. Momentum will cause opposing components of the step at successive positions to be canceled, and reinforcing components to be enhanced. This has many potential benefits. Acceleration across long regions of shallow but fairly constant gradient can be achieved. If sufficient momentum is held on entering a local minimum, this may allow uphill escape from the opposite side.

A graphical example of the effect of momentum in a two-dimensional weight space is shown in Fig. 4. A single node system with two input weights and a bias weight fixed at 3.9 was permitted to train on the function $\cos(x_1 + x_2)$. Both inputs were constrained in the training set to the range 0–$\pi/4$. This provided a simple two-dimensional error surface with a broad minimum. Full batch mode learning was used. Thirty-five steps were taken, beginning with an off-axis approach position. Fig. 4a shows progress with a stepsize of 2.0 and momentum of 0.0 and Fig. 4b shows progress made with the same stepsize and momentum of 0.9. In a given application, the actual settings of these parameters must be scaled according to the size of the features in error space, however, these plots provide a qualitative perspective. Although momentum here provided a somewhat circuitous route towards the minimum, it nonetheless allowed greater progress in this case, for the same number of steps. More complicated weight vector trajectories were observed at other settings, but this same qualitative argument was generally true.

3.8.4. Noise on the error surface

The error surface in practical problems can be 'noisy', rather than smooth. This may be due to errors in measuring or estimating the input and/or output vectors used in training. A detri-

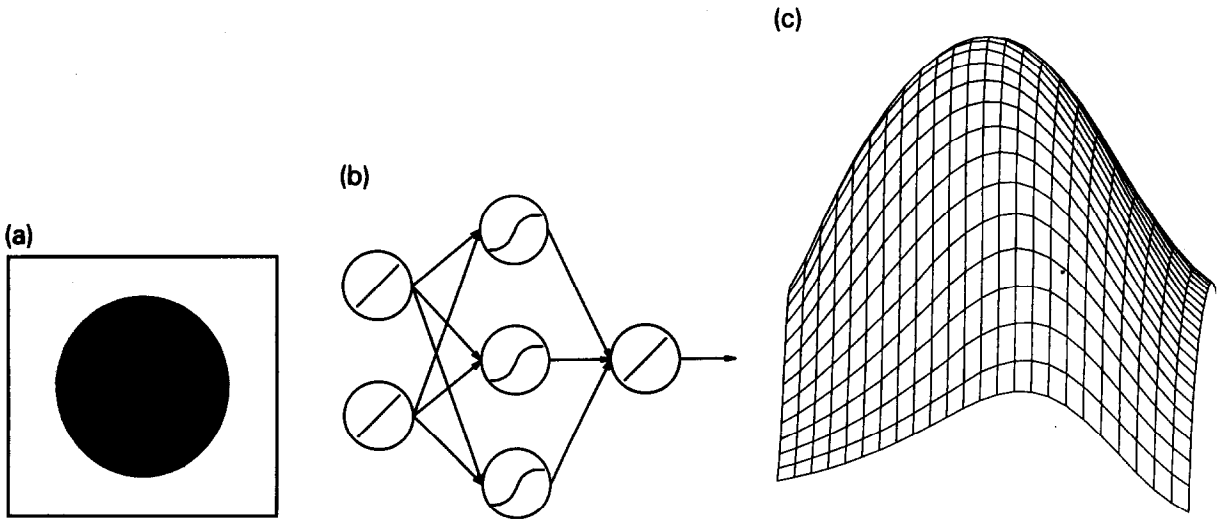


Fig. 8. A 2:3:1 network solution to the circle in a square problem, employing a linear output node. (a) Schematic representation of the problem; (b) schematic diagram of the network architecture; (c) surface plot of the fitted response surface in the two-dimensional input space; (d) contour plot of the fitted response surface.

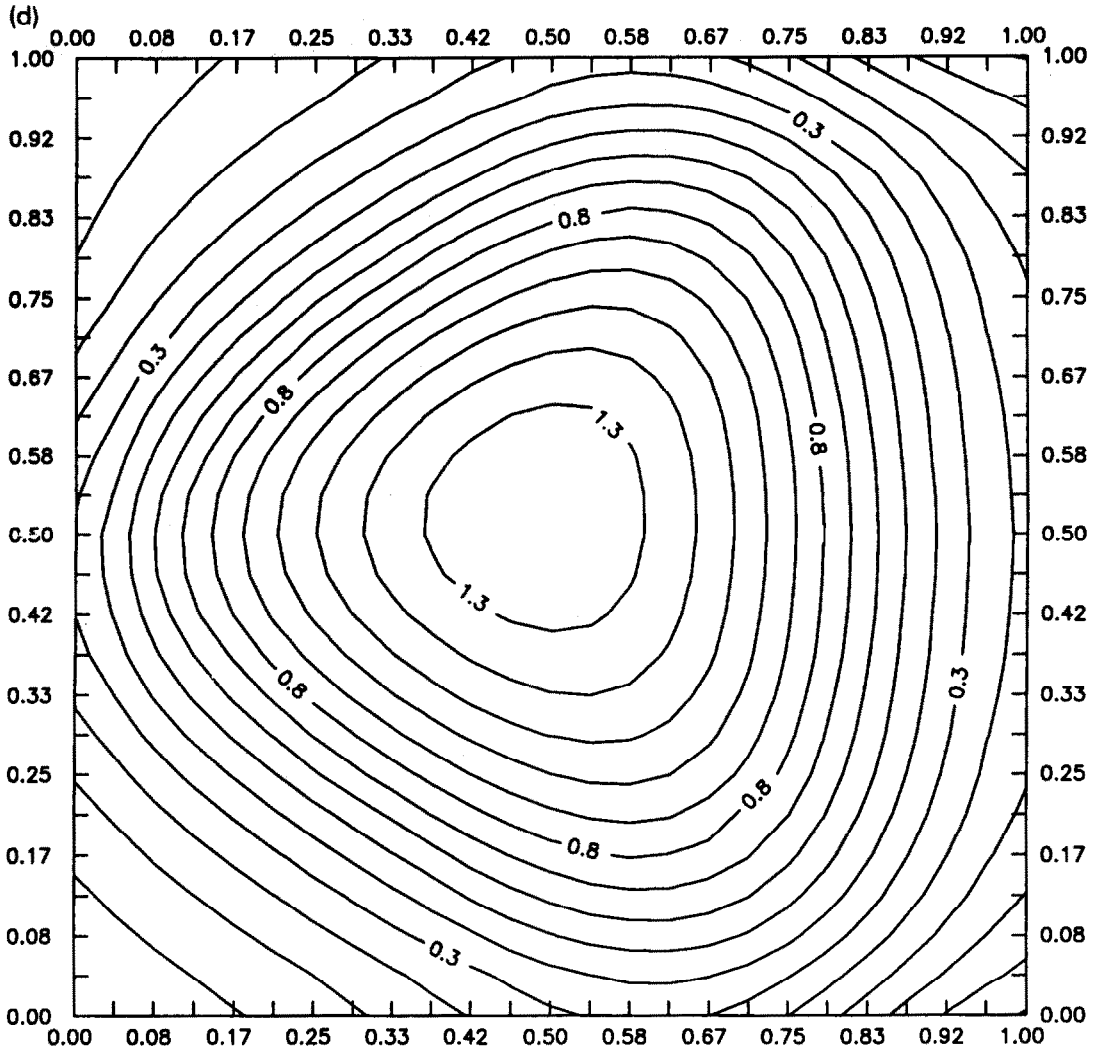


Fig. 8 (continued).

mental consequence is that location of a minimum is often delayed by superfluous random local gradients due to this noise. The use of a momentum term helps to filter out the effect of high frequency noise on an error surface. A possible beneficial effect of this noise, however, is that the superposition of a stochastic component on the search process results in a more thorough exploration of the error surface. This can often be sufficient to escape from shallow local minima, or circumvent others. For this reason, many workers purposely add independent noise to the input or output vectors, or randomly perturb the weights of the network at regular intervals during training. These perturbations would normally be decreased in magnitude as training progresses, to allow more accurate location of the precise minimum.

3.8.5. Setting the learning parameters

Unfortunately, there are no hard and fast rules for setting the learning parameters commonly used: the stepsize and momentum terms. While smaller settings will produce more ideal behavior,

they will also exacerbate an already computationally expensive problem. Momentum, if used, must be less than 1.0 in order for learning to be stable. Otherwise the weight vector will move on the error surface under greater influence from the previous gradient than from the current gradient, for a given step. Typical settings are from 0.40 to 0.90.

Stepsize, on the other hand, can be set at any value — typical settings range from 0.10 to 10.0, if sigmoidal output nodes are employed. Error surfaces with broad shallow minima will be best searched with a larger stepsize, while those with small steep minima are best searched with a smaller value. Settings which are too large may promote oscillation about a minimum, or may cause a jump completely over a global minimum into an attractive basin of a local minimum.

3.8.6. Divergent weight oscillations

One behavior which may be observed during training is that of divergent oscillations in the weight vector position, eventually causing the weight values to 'explode'. This can be explained using a simple example: consider a one-dimensional error surface with a quadratic minimum:

$$E = k(w - w_{\min})^2 \tag{13}$$

The error gradient would therefore be given by:

$$\frac{\partial E}{\partial w} = 2k(w - w_{\min}) \tag{14}$$

and the gradient would therefore continually increase with the term $(w - w_{\min})$. If the learning parameters are set such that the first step taken carries the weight vector across the minimum to a distance farther from the minimum than the starting point, then the weight vector is already doomed to undergo divergent oscillations, as each step will cause a jump to the opposite side further than the previous one, encountering ever-steeper gradients, and so taking ever-larger steps back and forth (Fig. 5). The use of momentum will also discourage divergent oscillations in the weight vector, due to the cancellation of opposing step components, as discussed above.

This phenomenon is much more common in networks employing linear output nodes. Since

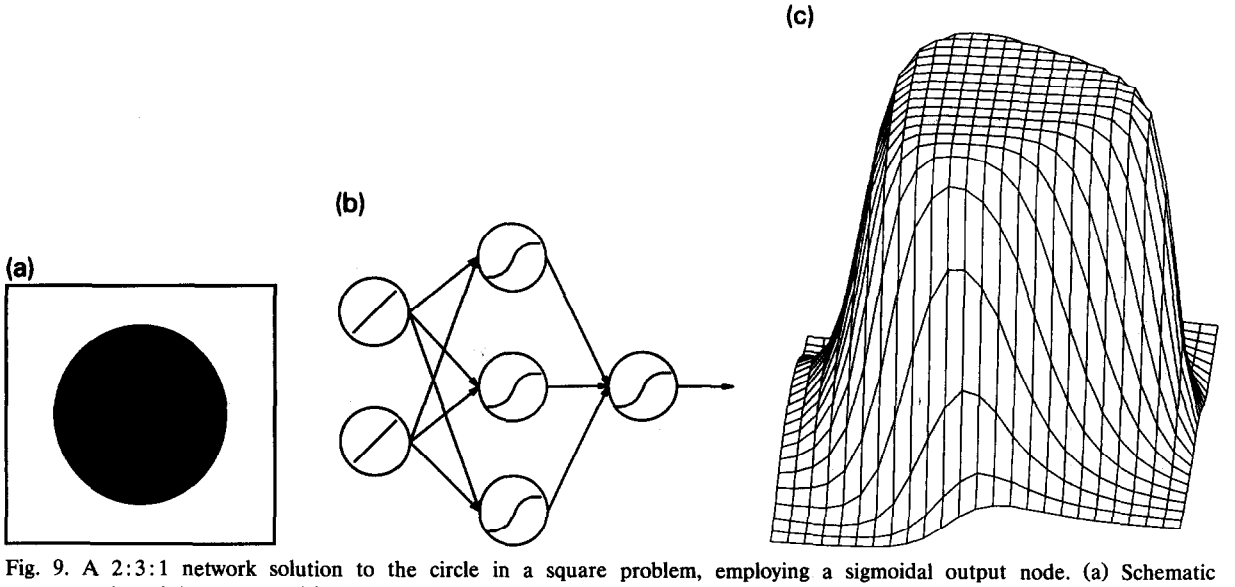


Fig. 9. A 2:3:1 network solution to the circle in a square problem, employing a sigmoidal output node. (a) Schematic representation of the problem; (b) schematic diagram of the network architecture; (c) surface plot of the fitted response surface in the two-dimensional input space; (d) contour plot of the fitted response surface.

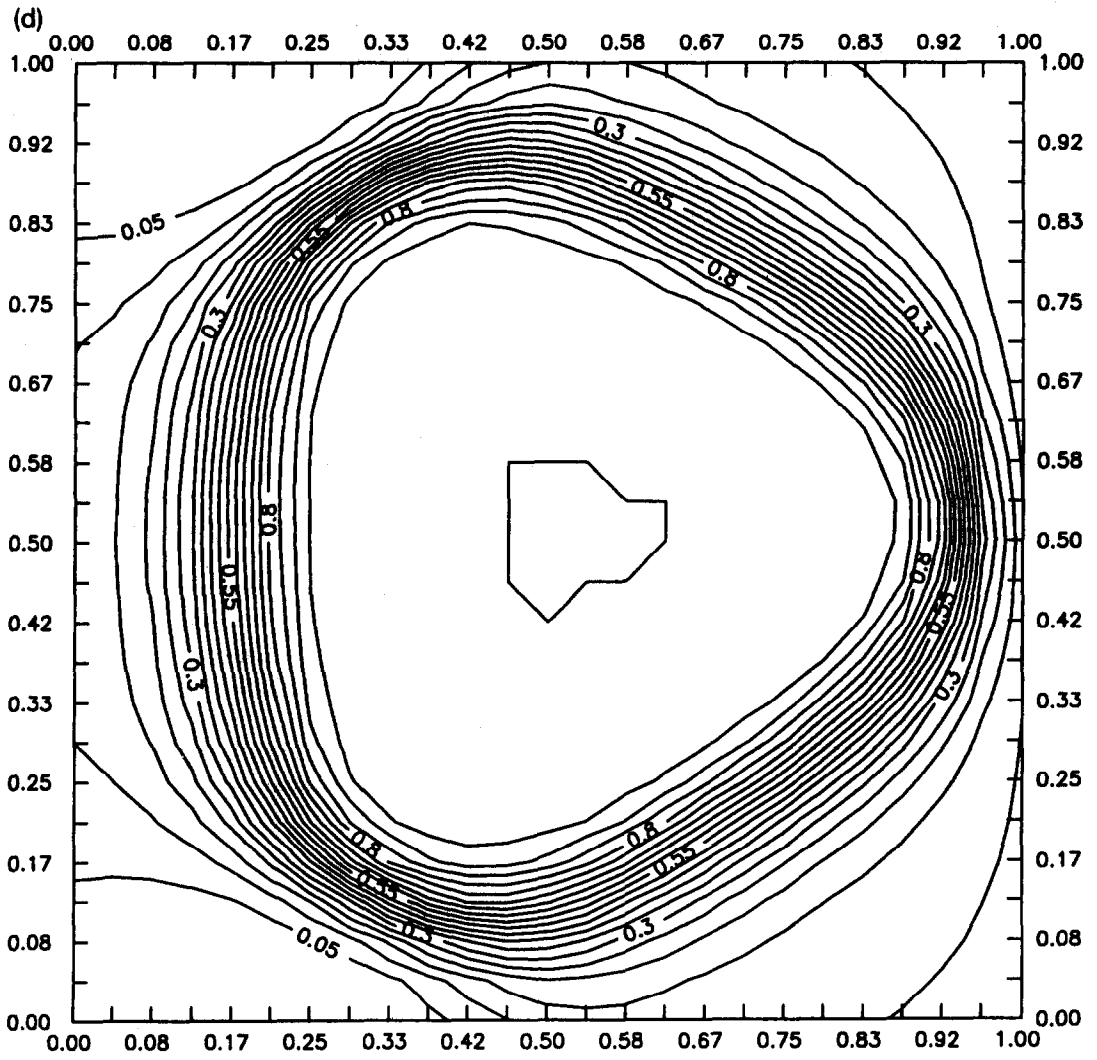


Fig. 9 (continued).

the output of such nodes is unbounded, the error surface in such a system must also be unbounded, rendering the situation described above more likely. When sigmoidal nodes are used at the outputs, however, the bounded output of these nodes places a corresponding bound on the error surface. Therefore, larger stepsize parameters may be typically employed in networks containing only sigmoidal nodes.

Weight values may also be caused to explode if the error surface decreases monotonically along a particular weight dimension. This situation can be avoided by performing explicit bounds checking on the weights during training.

4. MODIFICATIONS AND ALTERNATIVES TO BACKPROPAGATION LEARNING

4.1. Fahlman's modification

It is clear from Fig. 2a that the derivative of the sigmoidal transfer function approaches zero when the magnitude of a neuron's net input is

large, regardless of its polarity. This portion of the weight error derivative may inappropriately contribute to very small steps for a node when its weights create large net inputs for the majority of the input patterns. It has been suggested that many apparent local minima may actually be due to 'stuck units' in such a situation. A simple but effective solution was proposed by Fahlman [34]: add a small constant amount to the sigmoid derivative term in the weight error derivative, to avoid that contribution from approaching zero. The actual derivative of Eqn. 2 ranges from approximately zero to 0.25. Adding a value of 0.010–0.10 to this helps to avoid stuck units, and has been shown to provide more robust, and in many cases, faster learning.

4.2. Delta-Bar-Delta

There exist a number of variations on the so-called 'Delta-Bar-Delta' rule for the learning parameters [35]. In these methods, each weight is often given its own stepsize parameter which is varied as training proceeds. In theory, this frees the operator from worrying about setting the learning parameters, and lets feedback from the error surface itself be used to dynamically adjust them. The following algorithm states the essentials of the modified Delta-Bar-Delta algorithm of Tollenaere, named 'SuperSAB' [36]:

For each weight:

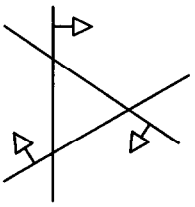
(1) If a step preserves the sign of the gradient from the previous step, increase the stepsize by a factor of *StepIncr*, where *StepIncr* is > 1.0.

(2) If a step inverts the sign of the gradient, kill any existing momentum, undo the step, and decrease the stepsize by a factor *StepDecr*, where *StepDecr* is < 1.0 and should also be less than $1.0/StepIncr$, to provide stability.

Tollenaere suggests values of 1.05 and 0.50 for *StepIncr* and *StepDecr*, respectively. SuperSAB learning should not be combined with pure incremental learning, since the gradient estimates obtained from single patterns will cause such frequent sign inversions that no progress will be made. It is often wise with the Delta-Bar-Delta variants to perform explicit bounds checking of the weight values to ensure that they do not explode. While these methods may decrease learning times by an order of magnitude or more in some cases, it is important to note that the algorithm described above will prevent escape from some local minima, since a move across a valley and landing on the opposite side is undone. Nevertheless, such methods may be used in strategies to quickly locate a minimum, and then the optimization process restarted from a different position on the error surface. Due to the occurrence of local minima, it is always prudent to repeat a training procedure several times with any training set, to provide some assurance that a global optimum has been found.

4.3. Classical optimization methods

Eqns. 8–10 provide us with a measure of the partial derivative of the error surface with respect to each weight. Therefore, we can combine these equations with classical optimization methods applying gradient information, such as steepest descent, and conjugate gradient methods [37]. Steepest descent involves line minimization along an estimated gradient of the error surface. Once



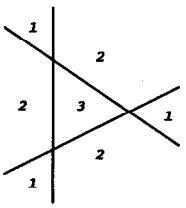


Fig. 10. Schematic diagram of (top) the disposition and (bottom) the integer summed output of the hidden nodes for the 2:3:1 network solution to the circle in a square problem.

this has been done, the new direction of line minimization is obtained from the gradient measured at the current minimum. Conjugate gradient methods represent a refinement to steepest descent, where a new search direction is chosen using second order information on the error surface. This greatly reduces the likelihood of 'backtracking' across a valley while moving toward the optimum, often speeding convergence significantly.

Alternatively, the information on the height of the error surface alone (Eqn. 4) may be used in conjunction with methods such as geometric simplex optimization [38], which has often been used

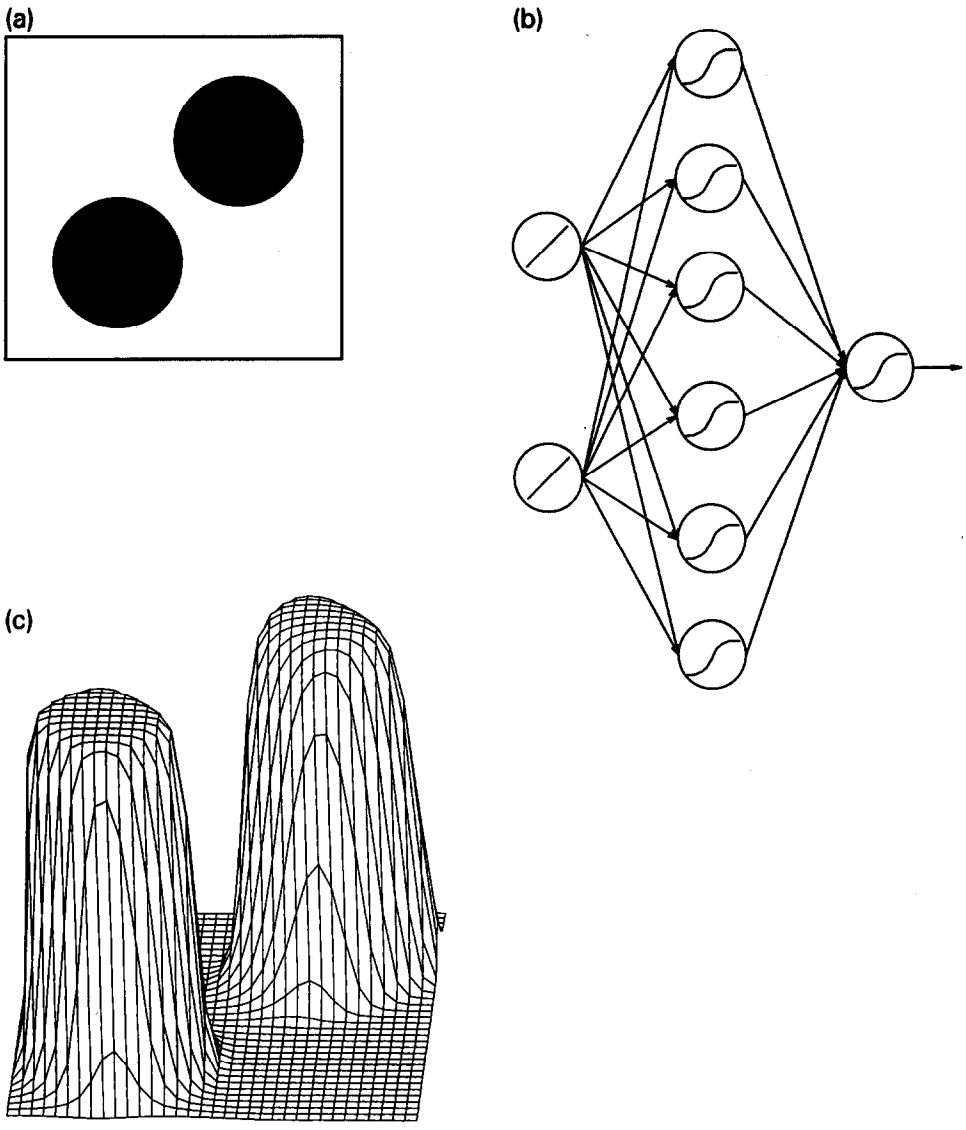


Fig. 11. A 2:6:1 network solution to the disjoint circles in a square problem, representing a single class modeling problem with two disjoint regions. (a) Schematic representation of the problem; (b) schematic diagram of the network architecture; (c) surface plot of the fitted response surface in the two-dimensional input space; (d) contour plot of the fitted response surface; (e) schematic diagram of the disposition of the hidden nodes.

in analytical chemistry. Another method using only the height of the error surface at a given point for control is simulated annealing [39]. All methods which make use of the height of the error surface obviously require Eqn. 4 to be evaluated over the entire training set.

Simulated annealing employs stochastic learning to achieve more robust convergence to the global optimum weights. The algorithm takes each step in a completely random direction. The error surface is then sampled, and if the error is lower, the step is accepted. Steps leading to higher error are accepted or rejected according to a probabilistic criterion in a manner analogous to 'Metropolis Sampling':

$$Prob(acceptance) = e^{-\Delta E/T} \quad (15)$$

where ΔE is the change in the error associated with the step, and T is the system temperature.

A uniformly distributed random deviate in the range zero to one is drawn and compared to the result of Eqn. 15. If the random value is less than the result, then the step is accepted. The innovation of simulated annealing is to provide a gradu-

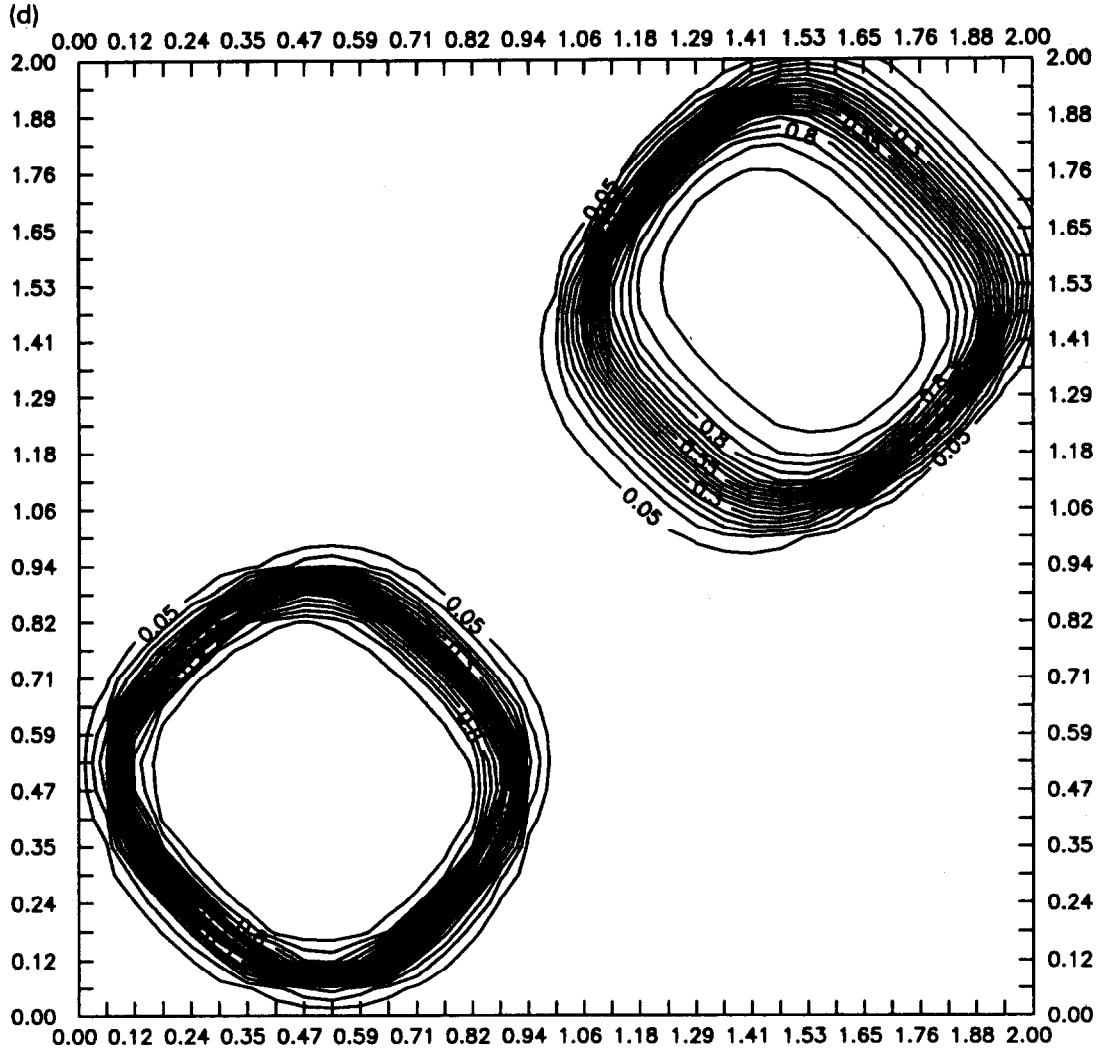


Fig. 11 (continued).

(e)

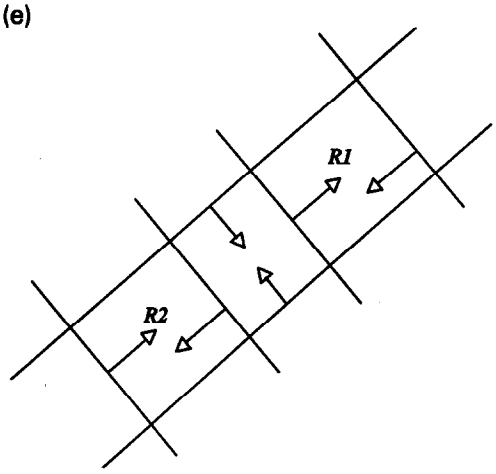


Fig. 11 (continued).

ally decreasing temperature parameter. The search process is started at high temperature, allowing free access to uphill steps. The temperature is gradually decreased, causing large uphill steps to become less likely. If the temperature is decreased slowly enough and the system is 'equilibrated' at each stage, then the system will in theory be located in the absolute and global minimum at zero temperature.

Although simulated annealing is seductive for its theoretical robustness, there are a number of difficulties with it. There is no 'correct' scheme for the temperature schedule, although an exponential decay of temperature is often used. In order to remain at true equilibrium, the temperature should decay infinitely slowly, and an infinite number of steps should be taken at each stage. This is obviously impractical. One of the greatest disadvantages of this method is its computational cost, which often requires current mainframe or even supercomputer technology for implementation on any but the smallest networks.

An interesting refinement to simulated annealing was reported by Xu and Oja [40]. While not referring to it by the common name, they combined simulated annealing with the so-called 'pocket algorithm'. The pocket algorithm is useful in conjunction with any optimization method wherein the quantity to be optimized is not a monotonic function of search time. This simple but effective modification requires that the best system state observed so far is always to be saved in our pocket (separate computer memory from the current system storage). On terminating the optimization process, then, the optimal system is pulled out of the pocket for use. Xu and Oja returned the system to this best state before restarting the search at each temperature step.

Bohachevsky et al. [41] reported another variation designed to make simulated annealing faster and more robust. They modified the exponential to include an additional term as follows (the form of the equation used above is retained):

$$Prob(acceptance) = \exp\left\{-\left[(E - E_{\min})^{g}\Delta E\right]/T\right\} \quad (16)$$

where E is the value of the error at the current location; $E_{\min}$ is the global minimum error; and g is a constant $\leqslant 0$. Basically, this added term makes the probabilty of accepting uphill steps smaller as the current height of the error surface decreases. Therefore, the deeper the minimum we are located in, the smaller the likelihood of escaping with this algorithm. As noted by Bohachevsky et al., a difficulty often associated with practical problems is that $E_{\min}$ may be unknown. They suggested that an initial estimate of $E_{\min}$ that is updated with information from the surface is often sufficient to provide fairly robust convergence, however.

The two modifications to simulated annealing just discussed could, of course, be combined to provide even greater confidence in the robustness of the method. It is important to recognize, however, that for practical multidimensional problems in which the complexity and scale of the features on an energy surface may be unknown and heterogenous, and the height of the optimum is unknown, there is still no way to guarantee that we will settle into (be trapped in) the global minimum in a finite number of moves and temperature steps. These stochastic methods do however, improve our likelihood of locating the global minimum.

5. CLASSIFICATION RESPONSE SURFACES AND THE SIGMOID FUNCTION

5.1. Sample problems

This section will present several highly simplified classification problems in one- and two-dimensional input spaces, along with neural network solutions to these problems. These low-dimensional input spaces are necessary to allow plotting the fitted network response function. Examination of these problems and the solutions provided by various network architectures is extremely helpful in getting a qualitative understanding of neural network solutions. In all cases, a unique output node was assigned to each class, and the 'correct' output values used in training were 1.0 for class presence, and 0.0 for class absence.

As noted earlier, one can consider a neural network in any application to be a function approximation device, and, in the case of a backprop network, to provide a generalized least squares approximation to the desired mapping function. From this perspective, one can consider that in a two layered network, the hidden layer constructs a nonlinear basis for the desired function, according to the least squares criterion. Furthermore, the output node(s) generate the desired function by taking a linear combination of the basis formed at the hidden layer, and passing it through their transfer function, which, for a sigmoid, is also termed a 'squashing function'. The reason for the term squashing function will become clear later.

Readers who are more comfortable with logic than with multivariate algebra may find the following alternative interpretation of two layered sigmoidal networks more helpful. We will first make the simplification that the sigmoidal nodes operate as discrete step functions, rather than graded analog functions. For a classification problem, each hidden layer node can then be considered to independently implement a crisp logical dichotomy (a Boolean function) on the elements of the real Euclidean space forming its input domain. Each output node can be considered to poll the hidden layer nodes to arrive at its own output. Depending on the relation between its weights and its bias, an output node may take the intersection (logical AND, symbolized by $\wedge$) of the hidden node outputs, or the union (logical OR, symbolized by $\vee$) of the hidden node outputs. If an output node is using a negative weight for a particular hidden node, then we can consider it to be taking the negation (logical NOT, symbolized by $\neg$) of that hidden node's truth value. We will return to this conceptualization in one of the detailed example problems below.

For classification problems, we can consider that the basis formed by the hidden layer nodes will be a nonlinear transformation, often including an expansion to higher dimensionality, if the problem is not linearly separable in the original space. The output node for any given class still acts as a linear classifier (Fig. 3c), therefore, the problem presented to it must be linearly separable, in order for the network to succeed.

Consider the example shown in Figs. 6a–d. This represents a one-dimensional classification problem, for which the class was contained in the interval [4,8] along the input axis. The desired function therefore maps the input value to 1 whenever it is contained in this set, or else maps it to 0. This problem is not linearly separable, however, as we cannot draw a single boundary on this axis which can separate the class interior from its exterior.

A reasonable (but not optimal, or unique) solution to this problem for a 1:2:1 (one input, two hidden layer nodes, one output node) network was constructed heuristically 'by hand' using the Eqns. 1–3 and consulting Figs. 2a and c in the following manner: first the two hidden layer sigmoids were 'placed' at the input locations 4.0 and 8.0, and 'pointed' right and left, respectively, with steep slopes. This was done by solving Eqn. 3 for the ratio of offset/weight for x locations of 4.0 and 8.0 for the sigmoid centers. Sizable weights (to initiate a sharp response) of identical magnitude but opposing sign were then used on the hidden node inputs to provide the symmetrical closed region desired. The chosen weights were: hidden node 1: (weight = 10.125, offset = −45.0), hidden node 2: (weight = −10.125, offset = 81.0).

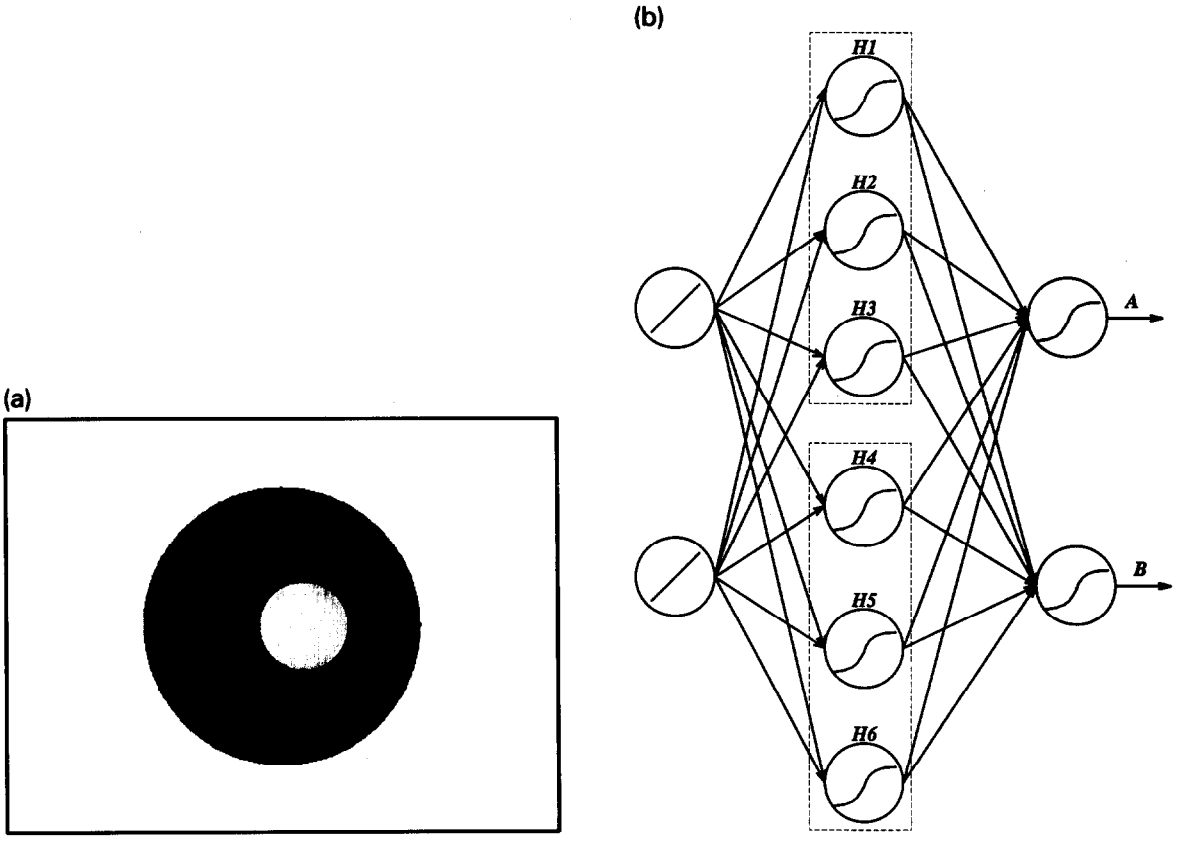


Fig. 12. A 2:6:1 solution to the embedded circles in a square problem, representing a two class problem, where one class (A) is embedded in the other (B). (a) Schematic representation of the problem; (b) schematic diagram of the network architecture; (c) surface plot of the heuristically generated response surface in the two-dimensional input space; (d) contour plot of the heuristically generated response surface; (e) surface plot of the fitted response surface in the two-dimensional input space; (f) contour plot of the fitted response surface; (g) discrete summed output of hidden nodes H1–H3 minus the discrete summed output of H4–H6.

For the output node, in order to produce values of 0 and 1 when outside and inside the class boundaries, the net input to the output node must then be a fairly large negative and positive number, respectively, when the external input is in these regions (consult Fig. 2a, for which the net input and external input values are identical). The values passed to it by the hidden nodes will be essentially (1, 0), (1, 1) and (0, 1), when on the left side, inside, and right side of the target area, respectively (Fig. 6b). Symmetrical positive input weights and a negative bias were then chosen for the output node consistent with these constraints. The chosen weights for the output node were: (weight 1 = 20.0, weight 2 = 20.0, offset = −30.0), chosen so as to produce a net input to the output node of −10.0 when outside the target region, and +10.0 when inside the target region.

The choice of solution weights was guided using the concept that the hidden layer node functions should be chosen so as to allow the required sharp, bump-shaped function response by linear summation and squashing at the output

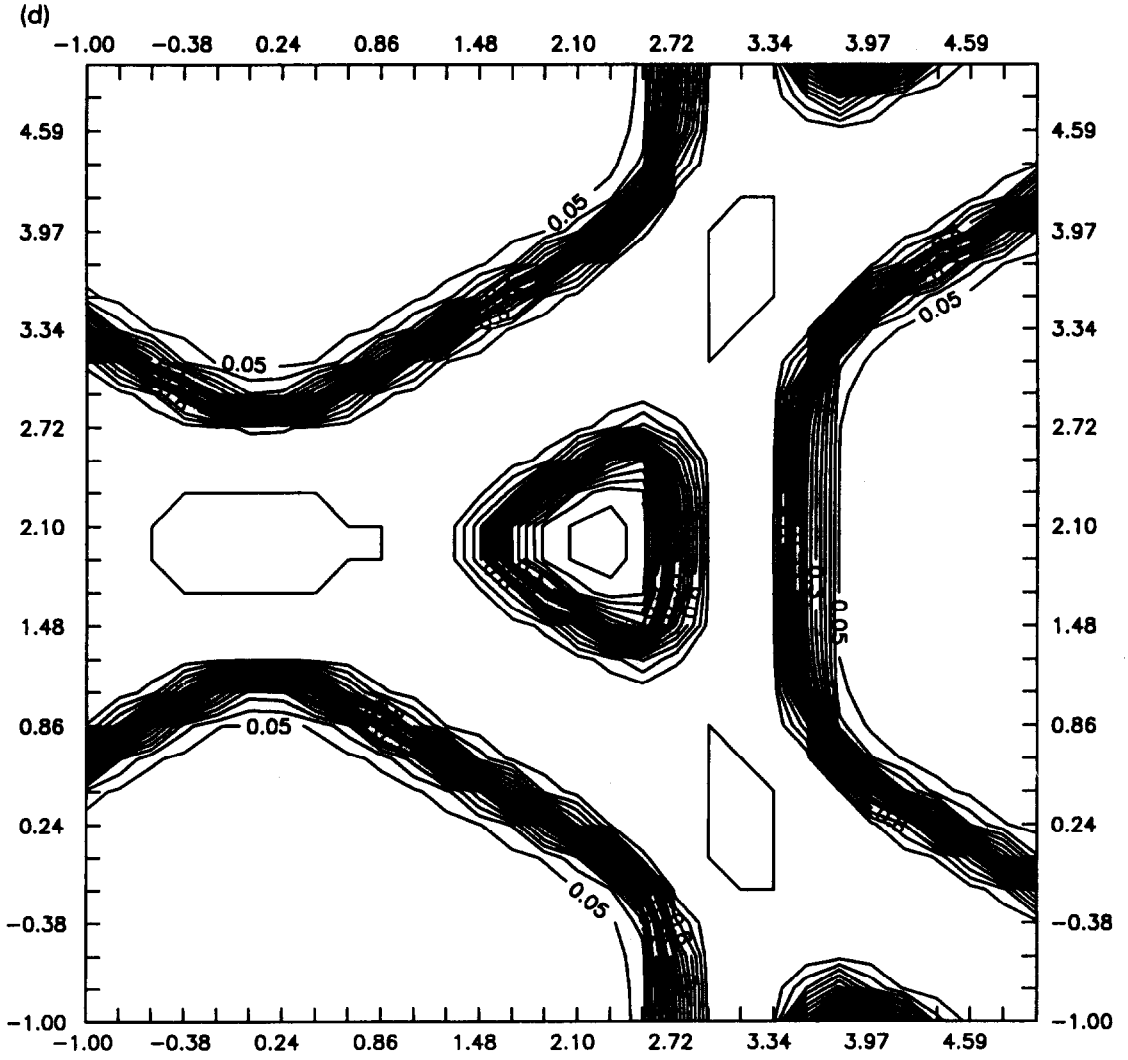


Fig. 12 (continued).

node. However, another helpful, and not incompatible view, is that in order to solve this problem, the hidden layer nodes must perform a nonlinear expansion of the input into a two-dimensional space, so as to transform the problem into one that is a linearly separable problem for the output node. Fig. 6d presents the problem 'from the perspective of the output node'. The points shown in Fig. 6d were generated by sampling the one-dimensional external input space over the interval [−20,20] at 100 evenly spaced locations. The output of the first hidden node was then plotted against the output of the second hidden node. The points are clustered at the locations ⟨0, 1⟩, ⟨1, 1⟩ and ⟨1, 0⟩, with the majority overlapping at these precise coordinates. This can of course also be deduced from Fig. 6b, but Fig. 6d clearly shows that the hidden nodes transform a one-dimensional nonlinearly separable problem into a two-dimensional linearly separable problem, for the output node. The linear decision boundary corresponding to the contour along which the output node response is 0.50 has also been sketched in Fig. 6d.

A more complex two-dimensional problem involving a single class, with a circular boundary shape, was considered next. The networks were asked to produce a response of one when the input vector was located inside a circle of radius 0.40 that was centered at 0.50, 0.50, and zero when the input vector came from outside this region. The ideal response function would therefore be a right circular cylinder of radius 0.40 centered at 0.50, 0.50, and with a perfectly flat top of height 1.0. We will call this the 'circle in a square' problem.

The first network solution employed a 2:2:1 architecture, employing all sigmoidal nodes (Fig. 7). Comparison of the response surface shown in Fig. 7 with the two-dimensional sigmoid shown in Fig. 3b indicates that the two sigmoids in the hidden layer are oriented so as to 'point' directly at one another, generating the parallel ridge response shown in Fig. 7. Refer also to Fig. 3c to convince yourself that the two hidden nodes must have parallel weights with opposing polarities and nonzero offsets, and that the output node must also use essentially identical input weights on the results from the two hidden nodes, in order to generate the overall response shown in Fig. 7.

Three hidden layer nodes are required to form a closed region in a two-dimensional input space, and the solution provided by a 2:3:1 network employing sigmoidal hidden nodes and a linear output node is shown in Fig. 8. It can be seen from the contour plot that the hidden layer nodes have arranged themselves at 120° angles in the input space, the required symmetry for the solution. Although we have now formed the closed hill-shaped response region, the sides of the hill are not very steep, and the top is not very flat.

Another 2:3:1 network was trained on this problem, this time employing a sigmoidal transfer function at the output node (Fig. 9). The C_{3v} symmetry of the solution is more apparent in these response contours than in the previous example, but is in fact the same. The sides of the hill have now achieved the desired steepness, and the top is very flat. The reason that this solution is so much improved is that the bounded output of the sigmoidal output node allows great amplification of the response of the hidden layer nodes, providing steeper sides without exceeding a value of 1.0 in the center, and thereby also flattening (or 'squashing') the top. In addition, the output node sigmoid can be considered (loosely) to per-

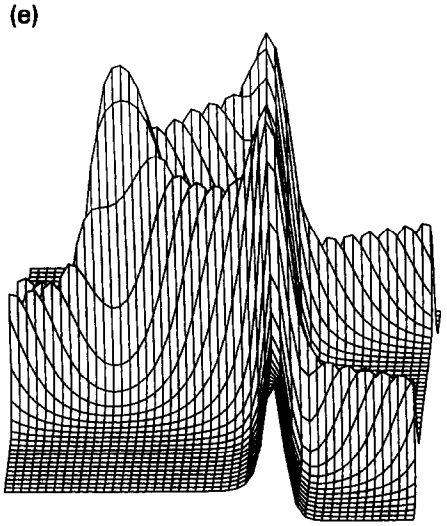


Fig. 12 (continued).

form a saturating high-pass amplitude filter function (Fig. 2a), and so the ripples which occur at the intersections of the 'feet' of the sigmoids are reduced as well. It may be curious to some readers that the hidden layer nodes are not amplified further, generating even steeper sides and a flatter top. While this could indeed be done, the curved regions at the intersection of the sigmoid regions of the hidden layer are due to the curved profile of the shoulder of the sigmoid function. Excessive amplification would square off the shoulder (see Fig. 2c), and produce a trigonal prism solution. The response function derived is the best compromise according to the least squares error criterion.

As noted above, the three sigmoids do not produce a perfect closed trigonal figure, but in fact have 'extra' raised regions formed by their intersection, to produce feet. The origin of these is made more clear with the schematic drawings of Fig. 10. Fig. 10a depicts the location and direction of the three sigmoids. Making the simplification that the sigmoids operate as step functions, Fig. 10b depicts the unweighted discrete

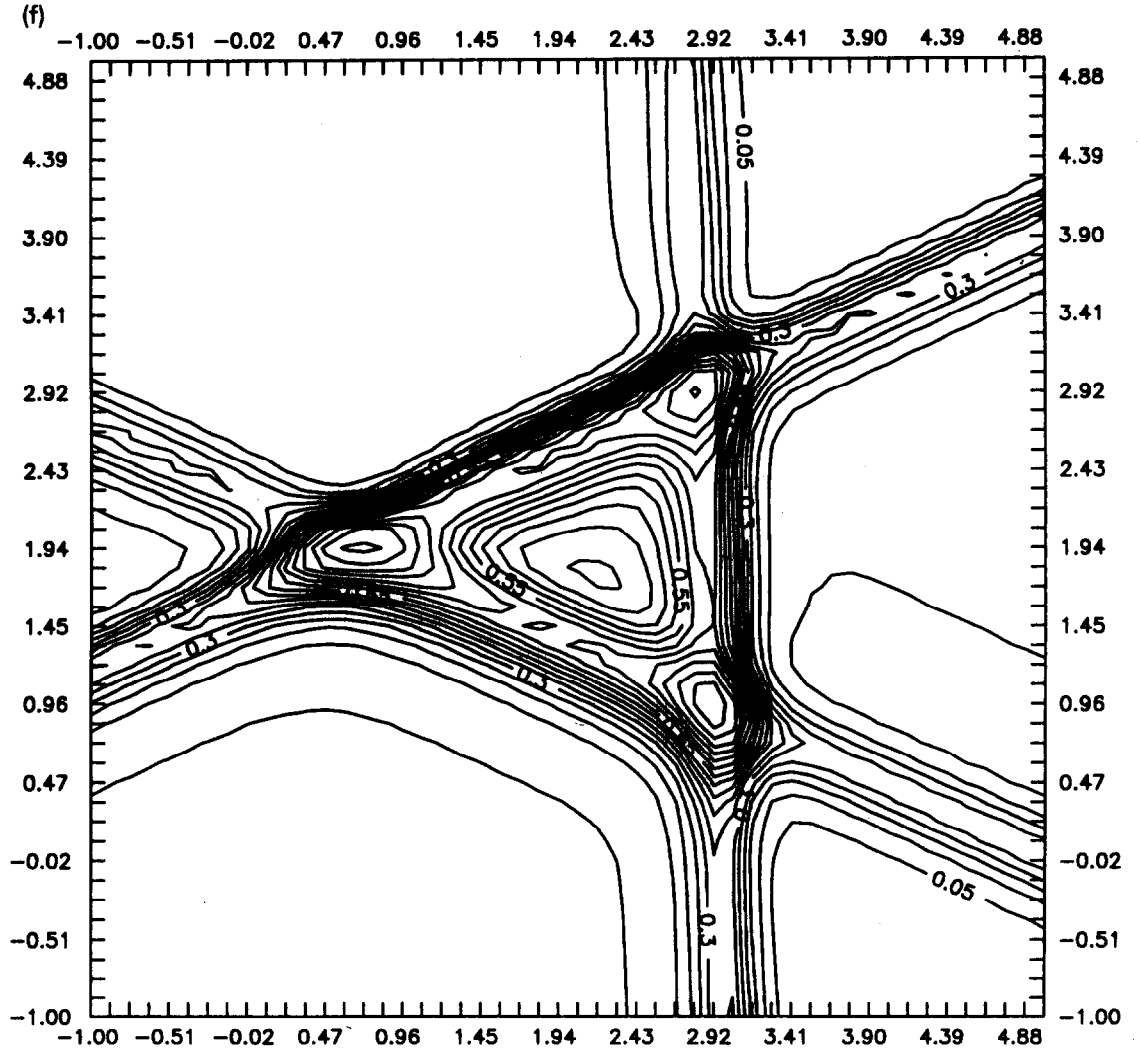


Fig. 12 (continued).

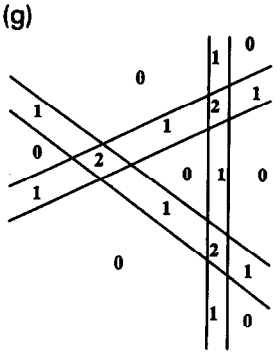


Fig. 12 (continued).

sum of their outputs, in the different regions of the input space. Of course the sigmoids are not step functions, but such a view simplifies a pictorial analysis. Clearly, there are three extra raised regions of summed amplitude two. These regions are still visible in the filtered output, as shown in Figs. 9c and d. Note that the thresholding 'front' of the output node transfer function has greatly reduced their relative magnitude, however. The affine transfer performed in generating the net input to the output serves to provide the bipolar range and the appropriate magnitude scale for the net input to produce output values of approximately 1 and 0 when inside and outside the class boundary, respectively.

The next problem posed was still a single class problem, but involved two disjoint circular regions for the class. A 2:6:1 network architecture was generated by extension of the solution of the first problem. It was initially thought that the solution would consist of two circular humps with C_3 symmetry, that is, that the six hidden layer nodes would partition into two groups of three to solve the problem according to a 'divide and conquer' approach. Instead, the solution showed two hills, but with four surfaces making up each hump (Fig. 11). The only way that this solution can be generated is that two sigmoids are used to generate the edges parallel to the major axis, two are used to generate the inside minor edges, and two to generate the outside minor edges. In other words, all six hidden nodes are being applied globally to the problem, with two being used 'twice' to form the major edges. The decision surfaces formed by the individual hidden nodes were examined individually to verify this conclusion, and are shown schematically in Fig. 11e.

The divide and conquer solution was later formed by starting the network from an appropriate set of weights, however the MSE for this latter approach, while low, was significantly higher than for the solution shown in Fig. 11c. It is certainly very reasonable that a globally constrained solution should be better than an (effectively) locally constrained piecewise solution, if symmetry in the desired function is exploited; and that, effectively, four sigmoids can generate a better approximation to a cylinder than three.

A concave region in a Euclidean space is one in which it is possible to connect two points in the region with a straight line segment that passes outside the boundary of the region. Despite early speculation that neural networks employing a single hidden layer of sigmoidal units could not form a concave decision region, this is now known to be within their capability. The final problem approached included such a concave set. In this problem, one circular class (class A) was completely embedded inside the other (class B). The outermost circular region was centered at 2.0, 2.0, and had a radius of 1.5. The inner class region was centered at 2.25, 2.00, and had a radius of 0.50. There was no overlap between the two sets. It was proposed intuitively that this problem could be minimally approached with the same number of hidden nodes as was used in the previous problem, six. Two output nodes were used for the two classes. Forming the shape of class B clearly posed the more difficult problem. The 'hole' within it corresponds exactly to the bounds of class A, however. Therefore, it was reasoned that by using six hidden nodes, three could be allocated to forming the outer circular boundary, and three to forming the inner circular boundary. The two different output nodes could simply use opposite signs on the weights to the hidden node groups, to generate the correct approximation to the desired shapes. In other words, the design called for one output node to generate a hump for the inner class A by summation of three sigmoid outputs, just as was done for the circle in a square problem. We will arbitrarily designate

these three hidden nodes as H1–H3. The other output node was intended to sum the three other hidden node outputs (H4–H6) to form the outer boundary of class *B*, and to subtract the output of H1–H3, in order to create the 'hole' in the interior, which corresponds to the region occupied by the class *A*. A schematic depiction of the problem and the initial network used to form the solution is shown in Figs. 12a and b. To revisit the crisp logic paradigm introduced earlier, the output node corresponding to class *A* was intended to form (H1 ∧ H2 ∧ H3). The output node corresponding to class *B* was intended to form (¬H1 ∧ ¬H2 ∧ ¬H3 ∧ H4 ∧ H5 ∧ H6). This numerical rule follows naturally when one seeks to answer the question: where is class *B* true? Class *B* is true inside the outer circle formed by hidden nodes H4–H6 (in the intersection of their true regions), and not inside the inner circle formed by hidden nodes H1–H3 (in the intersection of their true regions).

Numerous training runs from random starting positions failed to produce a solution with the desired concentric closed class boundaries. Speculating that this difficult problem might contain a global minimum that was difficult to approach using gradient descent, it was decided to write a simple computer procedure to assign the initial weight configuration. The goal in doing this was not to generate the optimal weights analytically, but simply to try to start the network optimization from a point within the attractive basin of the global minimum. The conceptual solution described above was encoded heuristically, together with the solution of the corresponding Eqn. 3, to generate the weights for the proper position and orientation of the hidden nodes. The hidden layer nodes were divided into two groups, one for each region. They were distributed symmetrically around the outer boundary of the respective circles and pointed inward. Opposite polarities were used on the weights connecting the hidden nodes to the output nodes, for the two regions. The response of the naive heuristic solution for the 2:6:2 network is shown in Figs. 12c and d. The networks were then trained as before, beginning from the heuristically assigned starting configurations.

The 2:6:2 net did find a stable solution to the problem from the heuristic starting weights, with concentric closed regions corresponding to the two classes. However, the error of the solution was still fairly large (Figs. 12e and f). The major problem appeared to be the excess lobes corresponding to the 'feet' of the overlapping sigmoids. The origin of these feet is shown more clearly in the schematic drawing of Fig. 12g. This drawing depicts the discrete unweighted sum of H4–H6 minus the discrete unweighted sum of H1–H3. In order to reduce the magnitude of the corresponding positive errors, the minimization process reduced the amplitude of the outer shell, thereby necessarily introducing negative deviations from the ideal response on the outer 'cylinder' walls.

Next, a 2:30:2 net was trained on the same problem. The starting weights were again heuristically assigned. This time, hidden nodes H1–H15 were assigned to the task of forming the inner region, and nodes H16–30 to forming the outer

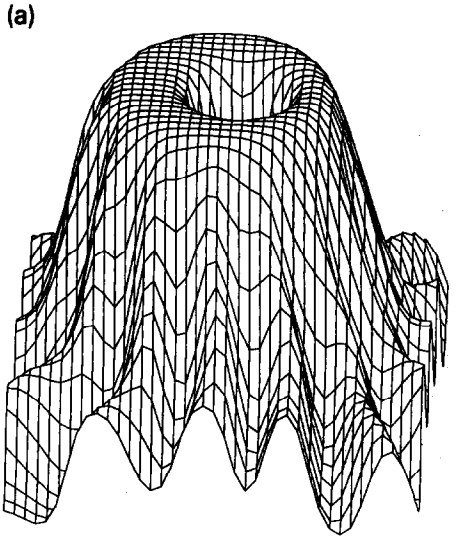


Fig. 13. A 2:30:2 solution to the embedded circles in a square problem. (a) Surface plot of the heuristically generated response surface in the two-dimensional input space; (b) contour plot of the heuristically generated response surface; (c) surface plot of the fitted response surface in the two-dimensional input space; (d) contour plot of the fitted response surface.

region. A much better solution, both qualitatively and quantitatively, was formed using 30 hidden nodes instead of six. The response surfaces for the naive heuristic starting configuration and the fitted network are shown in Fig. 13.

In order to provide a more detailed view of these two different solutions to the same problem, a program was written to provide plots of the discrete summed responses of the trained hidden nodes in the input space. The input space was scanned in raster fashion, and the hidden nodes outputs were summed at each position according to the following rule: if the output of the hidden node was greater than 0.50 (the midpoint of its range), then 1 was added to the summed output at that position, else 0 was added. The hidden nodes were divided into two groups according to the polarity of the weight between them and the output node corresponding to class *B*. Two plots were generated for each network, with one plot corresponding to sums of positively weighted hidden nodes, and one for negatively weighted nodes. The intensity values were scaled to span a 64-level grayscale. These plots for the

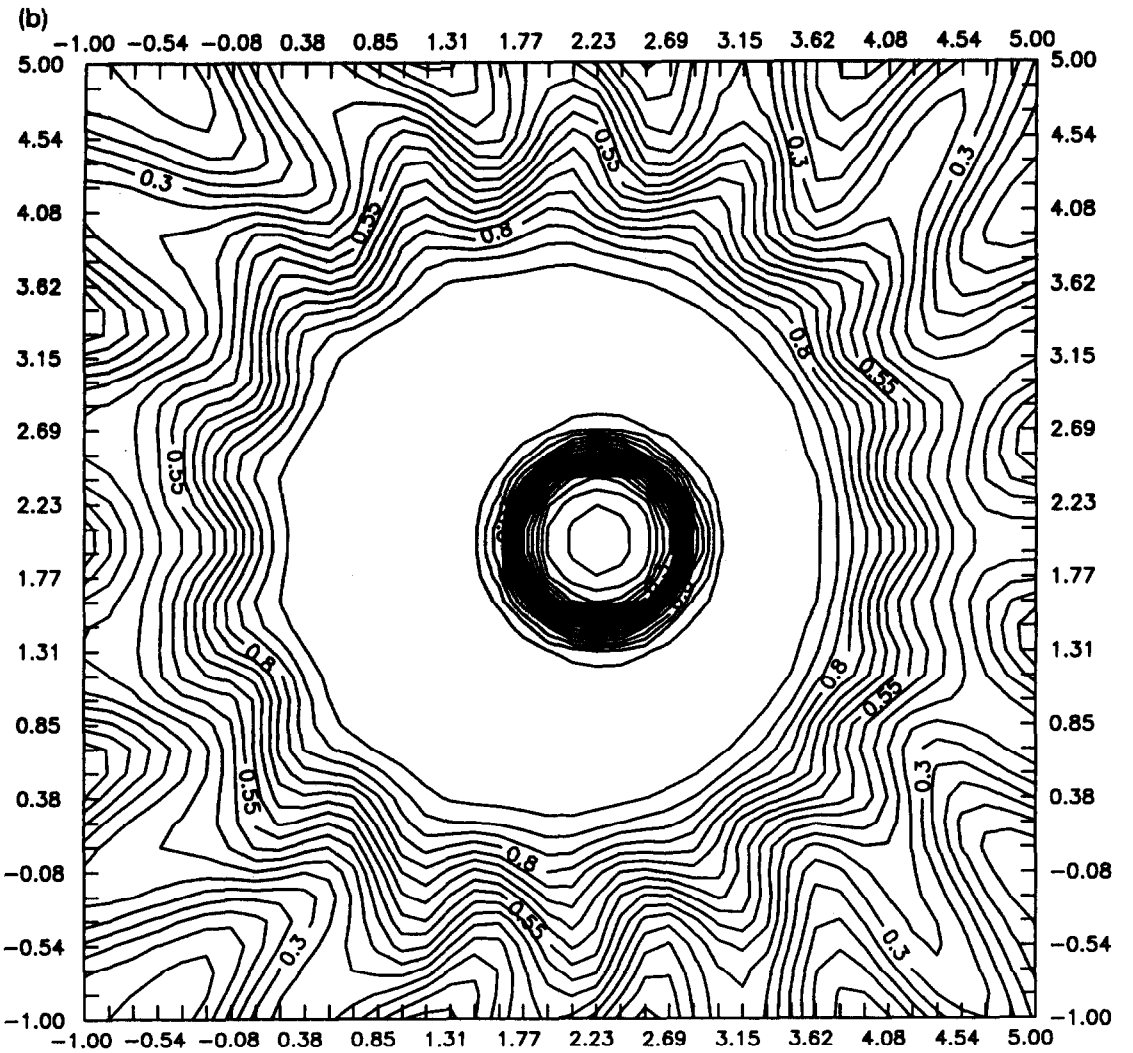


Fig. 13 (continued).

2:6:2 net are shown in Figs. 14a and b, and for the 2:30:2 net in Figs. 14c and d.

The ideal symmetries of the two target regions corresponding to class *A* and class *B* are both $C_{\infty v}$. The heuristic starting weights provided ideal symmetries of C_{3v} and C_{15v}, for the 2:6:2 and the 2:30:2 net, respectively. These symmetries were largely retained in the 'learned' solutions. One reason that the 2:30:2 net provides a better solution is that it has a closer approximation to the ideal symmetry. This can of course be predicted intuitively, and is shown to be true in Figs. 14c and d. In addition to this better rotational symmetry match, we find a second effect visible in these figures, which is actually more noticable in a color-mapped plot. The 'feet' generated by the 2:30:2 network hidden node overlaps decay more quickly as we move away from the region edge than those generated by the 2:6:2 net. This too provides a better approximation to the desired right circular cylinder shape for each class region.

5.2. Optimality of neural networks for classification

While classification has been performed with explicit modeling methods for many years, such methods can on reflection be seen to be inappropriate in many cases. For example, a number of explicit methods make use of a multivariate normal distribution fitted to the members of each class. While this approach may give the appearance of rigor, it should be noted that the multivariate normal distribution is intended to describe stochastic variation in a quantity. The different positions of class members in the input space are normally not primarily due to random errors in determining their location. Rather, they are the result of some underlying complex and deterministic processes which give rise to real differences in measured properties. There is no reason to expect that such variation will necessarily manifest itself as a multivariate normal distribution. Therefore, although a multivariate normal approximation of class shape may be useful, we can expect less constrained, self-modeling methods such as are implemented in neural networks to provide better solutions.

6. PROBLEMATIC ISSUES FOR BACKPROPAGATION NETWORKS

6.1. Overtraining and the frequency response of sigmoidal nodes

It is known that backpropagation networks which are 'too large' often pass through a point of maximum generalization during training, to eventually overfit the data. The usual approach to avoiding such behavior is to monitor the MSE on an independent cross-validation set during training as well. The training process is then stopped when the error on the cross-validation set is at a minimum. Chauvin [42] noted that networks exhibiting such behavior appear to start out by mapping the DC component of the input data, and move steadily toward mapping higher and higher frequency components. He rationalized this by pointing out that the maximum reduction in mean squared error on the training data is obtained by mapping the DC component, with successively smaller reductions obtained at higher and higher frequencies. This causes us to wonder how the network would 'know' this, and how such a frequency behavior can be explained in terms of the weight vector trajectory.

It is proposed here that the observed progression of frequency mapping is produced by the

(c)

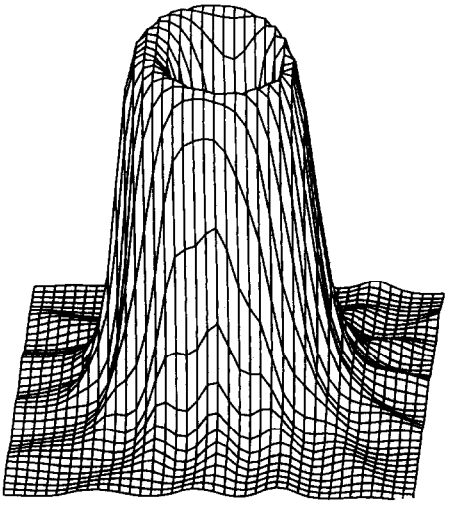


Fig. 13 (continued).

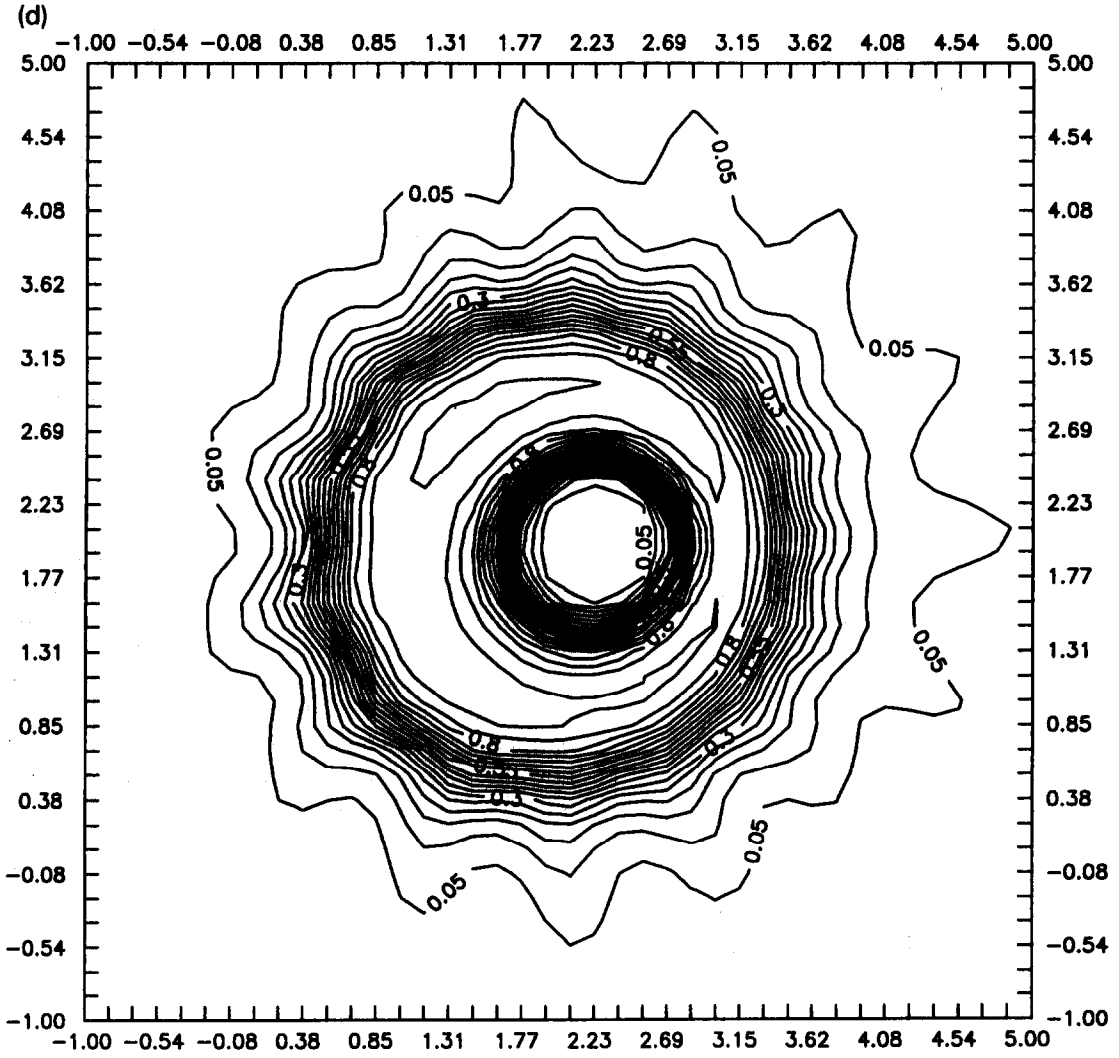


Fig. 13 (continued).

normal initial conditions (all weights set to small random values), and that the frequency progression is caused by the variation in the node response 'frequencies' as they move away from the origin to the minimum MSE. The overall frequency reponse of the system is directly related to the frequency response of its components, the individual neurons.

A 'frequency' could be ascribed to each node by determining the change in input required to produce some fixed macroscopic change in output. The corresponding change in input could than be termed a 'decay time' for the node, and inverted to produce a frequency. To simplify our analysis in multidimensional input space, we will instead consider the frequency of a given node to be equal to the maximum slope of its response curve, measured along the gradient of its response surface.

Proceeding in a manner analogous to that reported by McClelland and Rumelhart for deriving the partial derivative of the error with respect to a node weight [29], we can arrive at an expression for the partial derivative of a sigmoid node

output, with respect to an input dimension, obtaining after numerous steps:

$$\frac{\partial o}{\partial x_i} = -o(1-o)w_i = uw_i \tag{17}$$

Noting that the overall gradient can be obtained by summing the individual orthogonal components, and that the length of this net gradient vector can be obtained by normal Euclidean means, we obtain:

$$\sqrt{\sum_{i=1}^{n}\left(\frac{\partial o}{\partial x_i}\right)^2} = \sqrt{\sum_{i=1}^{n}(uw_i)^2} = u\sqrt{\sum w_i^2} \tag{18}$$

$$u\sqrt{\sum w_i^2} = -o(1-o)\sqrt{\sum w_i^2} \tag{19}$$

The expression $-o(1-o)$ is equal to its maximum magnitude of -0.25 at $o = 0.50$, therefore the magnitude of the maximum node slope (its 'frequency') is proportional to one-quarter the length of its weight vector. The frequency response behavior of a given node with respect to its weight space is therefore a radially symmetric 'hyper-cone' centered at the origin, where the frequency is 0.0.

It is perfectly reasonable therefore, that the overall dynamic frequency behavior of the network during learning is actually due to the initial conditions imposed for the error minimization, which, being located near the origin of the weight space, must produce a low frequency response. Moving away from the origin toward the solution will then necessarily produce higher frequencies in the network basis set (its hidden nodes).

In addition to providing some insight into generalization dynamics of oversized networks, this analysis may allow one to prevent overfitting, by

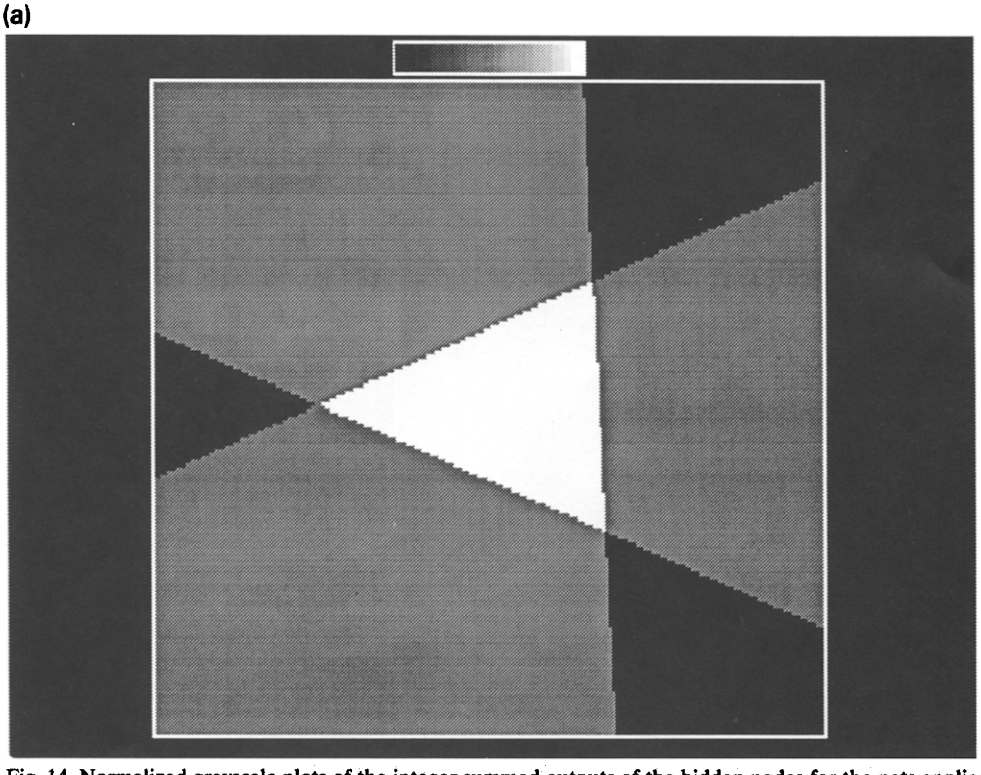


Fig. 14. Normalized grayscale plots of the integer summed outputs of the hidden nodes for the nets applied to the embedded circles in a square problem. (a) Positively weighted nodes for the 2:6:2 net; (b) negatively weighted nodes for the 2:6:2 net; (c) positively weighted nodes for the 2:30:2 net; (d) negatively weighted nodes for the 2:30:2 net.

limiting the frequency response (weight vector lengths) of the network nodes to match the frequency limit in the desired function to be approximated. It might also be possible to speed training by assigning the initial weights according to a frequency spectrum dictated by the desired response function.

6.2. Architecture and design

A difficult issue for the neural network engineer is deciding on the appropriate architecture for a given problem. Since response functions for quantitative analysis tend to be relatively simple (although not necessarily linear), this is not such a problematic limitation there. In qualitative analysis, however, we are essentially trying to fit class probability distributions in a multivariate input space. The true shape of such functions can have an absolutely arbitrary complexity. Even with a good understanding of how sigmoidal functions combine to generate such surfaces, we are limited in the multivariate case by the fact that we cannot even see the appearance of the desired function.

There has been a good deal of work aimed at generating 'optimal' network architectures for backpropagation networks employing sigmoidal nodes. Many can be divided into two categories: (a) beginning with a network which is too small, and systematically enlarging it by some scheme until the error on the training set and an independent test set drops to acceptable limits; (b) beginning with a network which is too large and systematically pruning connections and nodes, and observing the error to try to obtain a reduced system with acceptable error. The majority of such schemes are systematic and reasonable means of finding an economical solution to the 'appropriate architecture' question. However, such systems do not use any explicit information about the problem, and both types are essentially brute force approaches, requiring a great deal of

(b)

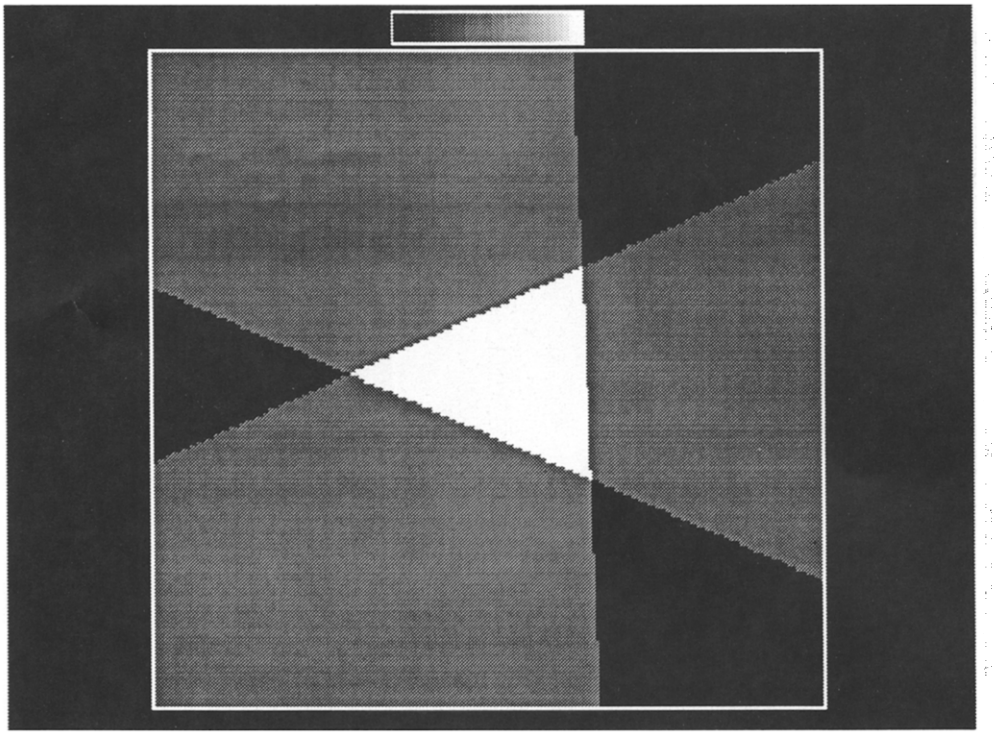


Fig. 14 (continued).

'wasted' computation along the path to the terminal architecture. Finally, the architectures derived are only optimal in the sense that they represent the best performing network encountered during what can be described as a heuristically directed search of an 'architecture space' wherein nodes in a search tree correspond to particular architectures. The morphology of the tree, as well as the entry point(s) are different for every algorithm.

Beyond such 'brute force' approaches, there have been several promising methods reported in the literature for learning systems which begin with an architecture which possesses too many hidden units, and dynamically prunes unnecessary weights and/or units as training proceeds. Two such methods, briefly presented here, add a second penalty term to the cost function (beyond mean squared error), to discourage the use of correlated weights or units.

Weigend et al. [43] described a method for pruning unnecessary weights by 'weight decay' according to the following cost function:

$$E = E_0 + \gamma \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{w_{ij}^2/w_0^2}{1 + w_{ij}^2/w_0^2} \tag{20}$$

where E_0 represents the MSE from Eqn. 4; w_{ij} represents the jth weight for the ith node; w_0 is a real constant; γ is a real constant; n is the number of training patterns; m is the number of output nodes.

This method was used by Curry and Rumelhart [13] in experiments concerning their mass spectrum interpretation network 'MSnet'. In addition, Chauvin [44] reported the use of a similar approach to extend the cost function by pruning unnecessary hidden units, one incarnation of which is presented here:

$$E = \mu_{er} E_0 + \mu_{en} \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{o_{ij}^2}{1 + o_{ij}^2} \tag{21}$$

(c)

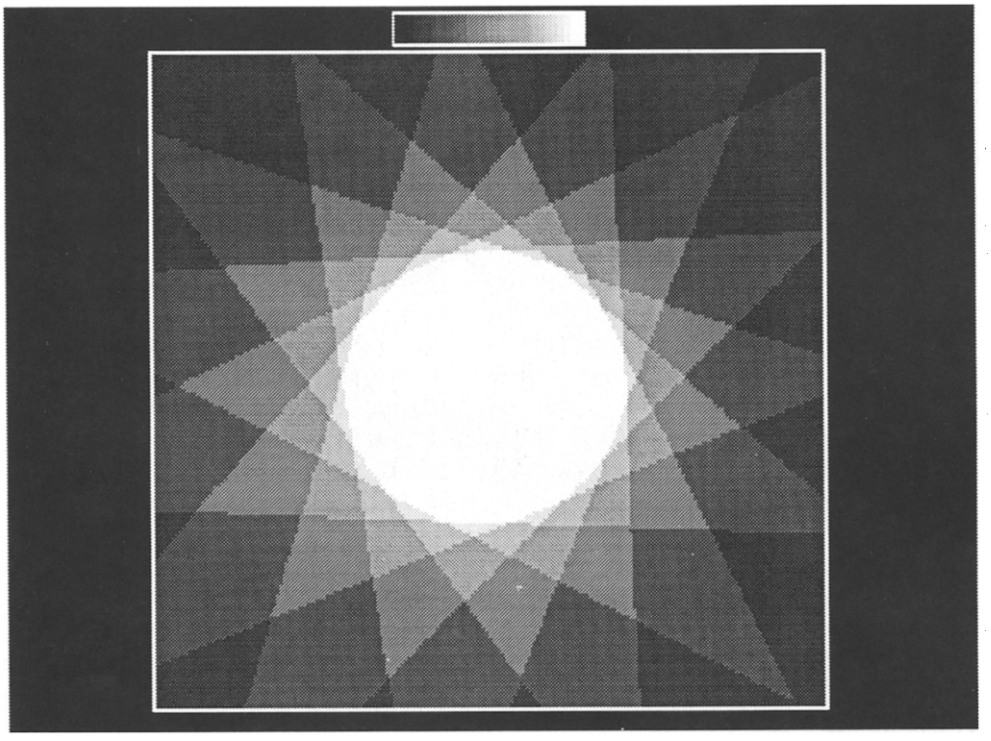


Fig. 14 (continued).

(d)

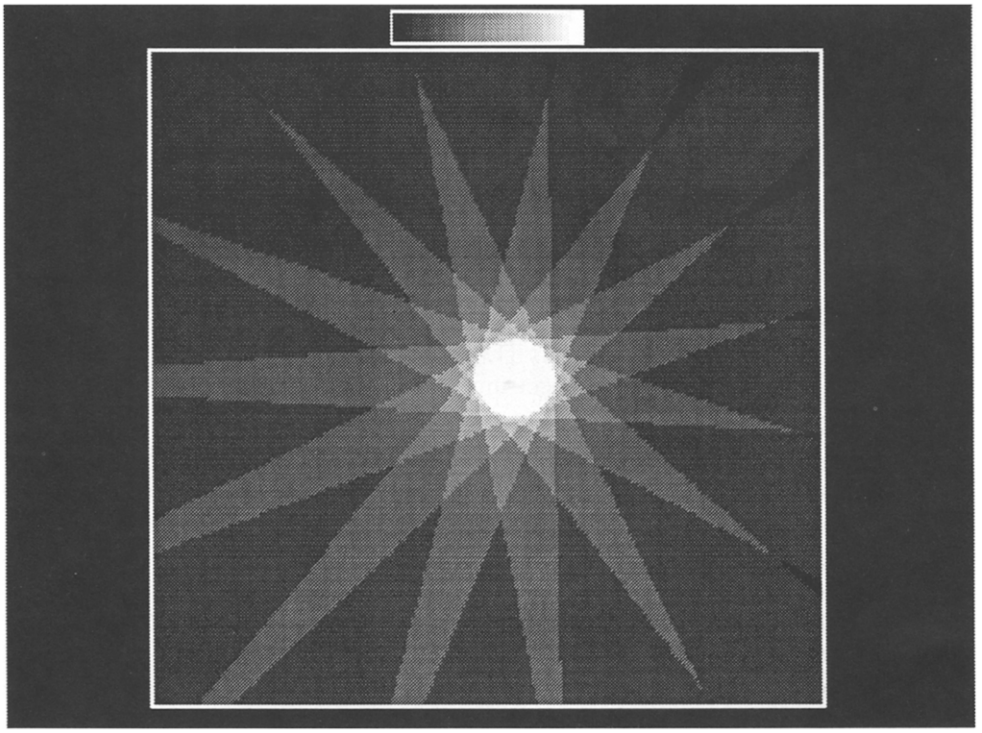


Fig. 14 (continued).

where E_0 represents the MSE from Eqn. 4; o_{ij} represents the output of the ith hidden node on the jth pattern; μ_{er} is a real constant; μ_{en} is a real constant; n is the number of training patterns; and m is the number of hidden nodes.

This version of Chauvin's method is obviously very similar in its form, if not meaning, to the former expression for minimizing redundant weights. Both expressions utilize some factor to adjust the relative importance of the MSE term, and the 'network complexity' term. Both of these equations penalize the system for employing excessive complexity, either in the number of weights or the number of hidden units used to achieve a given error. In doing so, they encourage the elimination of correlated or unimportant network components. It seems clear that aggressive use of either augmenting term too early in the training process can jeopardize convergence. Weigend and coworkers, in fact, suggested that the proportion of the cost attributed to weight magnitudes be held at zero initially, and increased in importance as training proceeded, while monitoring the MSE to prevent elimination of necessary weights. Such dynamic adjustment of the augmenting term may be tricky to perform in real applications. Nonetheless, these and related methods hold a good deal of promise for 'automatically' arriving at an approximately optimal architecture, and thereby achieving an efficient architecture in terms of storage space and computation time, as well as minimizing the likelihood of overfitting.

7. CONCLUSIONS

There is still a great deal of theoretical work which remains to be done in neural network research, which is one of the reasons that the

field is so exciting. The rather murky nature of neural network solutions may disturb some scientists, but the very problems which are appropriate are likewise ill-defined. What neural networks offer is the prospect of a usable solution to problems which cannot even be described analytically. While some claims of enthusiastic network proponents are no doubt exaggerated, these systems are not just a fad, and will in fact continue to increase in popularity and importance. The next ten years will show a remarkable number of solutions to ill-defined chemical problems.

One of the key problems facing neural network developers is training time. This is being attacked through research into more efficient learning algorithms, and with direct hardware support for neural network calculations. Some of these current hardware solutions claim to offer billions of backpropagation connection updates per second. Another important issue is convergence probability, and the hazards of local minima, i.e. understanding learning dynamics. These problems, too, are being attacked by current researchers. Finally, there is the extremely complex problem of deriving an 'optimal' architecture for a specific application. While important strides are being made on this subject, we should not expect this problem to be completely solved in the near future. The optimal architecture is related to both the functional complexity and the 'inherent dimensionality' of the desired mapping. Accurate assessment of these factors for arbitrary problems is an extraordinarily complex task.

Although we have concentrated here on backpropagation networks, there are new network architectures and learning paradigms introduced virtually every month in the research literature, along with new theoretical results elucidating the mathematical properties of existing systems. This is a field pregnant with advances, and the possible applications in chemistry are limited only by the imaginations of today's chemists.

For those readers who are unwilling or unable to develop their own neural network software, there is a large menu of commercial packages available that will run on most popular hardware platforms. The author has not reviewed, and does not endorse any of these commercial programs, however, a brief guide to such systems was published in the June, 1992, issue of *AI Expert* magazine.

8. GLOSSARY

Logic symbols:
- $\neg$: represents logical negation (NOT);
- $\wedge$: represents logical conjunction (AND);
- $\vee$: represents logical disjunction (OR).

Activation: Same as neuron output.

Affine function: A generalization of a linear function. In the geometrical sense, it can be considered to add translation capabilities to the rotation and scaling capabilties of a linear function.

Architecture: The topological arrangement of neurons, layers, and connections, which, in a complex way, defines the set of modeling equations available to the net.

Generalization: The ability of the network to produce the correct output for patterns not present in the training set, indicating induction of a valid mapping rule from the training set.

Gradient descent: Procedure for iteratively minimizing some quantity by moving in the $(-)$ direction of the gradient (derivative, slope) of the response from the current state of the system, (loosely) 'by always moving down the fall line'.

Hidden layer: Any of one or more layers normally used between the input and output layers in backpropagation networks. So called because it receives no input from, nor produces output to the 'outside world', i.e., it calculates only intermediate values used in determining the overall network output.

Input layer: The 'first' or 'lowest' layer of neurons, which is used only to broadcast the input values for the desired mapping to the first processing layer.

Input vector: The ordered set of inputs comprising the known or independent values for a pattern, e.g., spectral absorptions.

Mapping: The act of (verb) or rule for (noun) producing (an) output value(s) corresponding to a given input, e.g. $y = kx$ is a rule for mapping x to y.

Neuron (syns: node, processor, unit): The funda-

mental building block of a neural network. Normally, each neuron takes a weighted sum of its inputs to determine its net input. The net input is then processed through its transfer function to produce a single-valued output which is broadcast to 'downstream' neurons.

Net input: normally the inner (dot) product of a neuron's input vector with its weight vector.

Output layer: The 'last' or 'highest' layer of neurons, which is used to provide the output values required by the application.

Output vector: The ordered set of outputs comprising the unknown or dependent values for a pattern, e.g., component concentrations or identities.

Recall (syn: memorization): The ability of a network to reproduce the correct output values for the training set.

Sigmoid: An 'S-shaped' nonlinear transfer function commonly used in artficial neurons.

Supervised learning: A system, such as backpropagation, for deriving a rule for mapping inputs to outputs, i.e., for learning an approximation to a desired function. Supervised learning operates using known outputs corresponding to each input in the training set, and adapts the mapping rule to bring the approximation as close as possible to the desired (true) mapping as provided by the training examples.

Synapse (syn: connection): A connection between two nodes, with an associated weight. $Node_a$ passes its output to $node_b$ through such a synapse. The value received by $node_b$ is the product of the output of $node_a$ with the weight on this synapse.

Test set: A set of example input patterns, along with known correct output patterns, which is independent of the training set, to be used to evaluate the performance and validity of the learned mapping.

Training set: A set of example input patterns, along with known correct output patterns, to be used for learning.

Transfer function: The function a neuron uses to operate on its net input (e.g. sigmoid, linear, quadratic) to produce its activation level.

Vector: An ordered set of numbers. For an n-element vector v, the ith element can normally be considered to be the ith coordinate in n space, or the length of the projection of v along the ith axis.

ACKNOWLEDGMENTS

The author gratefully acknowledges the support of the National Research Council Postdoctoral Fellowship program while this manuscript was being prepared. This paper is an official contribution of the National Institute of Standards and Technology and therefore not subject to copyright in the United States.

REFERENCES

1 W.S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics*, 5 (1943) 115–133.
2 D.O. Hebb, *The Organization of Behavior*, Wiley, New York, 1949.
3 F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain, *Psychological Review*, 65 (1958) 386–408.
4 P.C. Jurs, B.R. Kowalski, T.L. Isenhour and C.N. Reilly, Computerized learning machines applied to chemical problems: investigation of convergence rate and predictive ability of adaptive binary pattern classifiers, *Analytical Chemistry*, 41 (1968) 690–695.
5 B.R. Kowalski, P.C. Jurs, T.L. Isenhour and C.N. Reilly, Computerized learning machines applied to chemical problems: interpretation of infrared spectrometry data, *Analytical Chemistry*, 41 (1969) 1945–1949.
6 D.R. Preuss and P.C. Jurs, Pattern recognition techniques applied to the interpretation of infrared spectra, *Analytical Chemistry*, 46 (1974) 520–525.
7 R.W. Liddell III and P.C. Jurs, Interpretation of infrared spectra using pattern recognition techniques, *Analytical Chemistry*, 46 (1974) 2126–2130.
8 M. Minsky and S. Papert, *Perceptrons*, MIT Press, Cambridge, MA, 1969.
9 J.U. Thomsen and B. Meyer, Pattern recognition of the ^1H NMR spectra of sugar alditols using a neural network, *Journal of Magnetic Resonance*, 84 (1989) 212–217.
10 E.W. Robb and M.E. Munk, A neural network approach to infrared spectrum interpretation, *Mikrochimica Acta*, I (1990) 131–155.
11 M.E. Munk, Neural network models for infrared spectral interpretation, *Mikrochimica Acta*, II (1991) 505–514.
12 B.J. Wythoff, S.E. Levine and S.A. Tomellini, Spectral peak verification and recognition using an artificial neural network, *Analytical Chemistry*, 62 (1990) 2702–2709.
13 B. Curry and D. Rumelhart, MSnet: A neural network

that classifies mass spectra, *Tetrahedron Computer Methodology*, 3 (1990) 313–237.
14 D.W. Elrod, G.M. Maggiora and R.G. Trenary, Applications of neural networks in chemistry. 1. Prediction of electrophilic aromatic substitution reactions, *Journal of Chemical Information and Computer Science*, 30 (1990) 477–484.
15 N. Qian and T.J. Sejnowski, Predicting the secondary structure of globular proteins using neural network models, *Journal of Molecular Biology*, 202 (1988) 865–884.
16 D.G. Kneller, F.E. Cohen and R. Langridge, Improvements in protein secondary structure prediction by an enhanced neural network, *Journal of Molecular Biology*, 214 (1990) 171–182.
17 J.R. Long, H.T. Mayfield, M.V. Henley and P.R. Kromann, Pattern recognition of jet fuel chromatographic data by artificial neural networks with back-propagation of error, *Analytical Chemistry*, 63 (1991) 1256–1261.
18 S.M. Chang, Y. Iwasaki, M. Suzuki, E. Tamiya, I. Karube and H. Muramatsu, Detection of odorants using an array of piezoelectric crystals and neural-network pattern recognition, *Analytica Chimica Acta*, 249 (1991) 323–329.
19 M. Glick and G.M. Hieftje, Classification of alloys with an artificial neural network and multivariate calibration of glow-discharge emission spectra, *Applied Spectroscopy*, 45 (1991) 1706–1716.
20 J.R. Long, V.G. Gregoriou and P.J. Gemperline, Spectroscopic calibration and quantitation using artificial neural networks, *Analytical Chemistry*, 62 (1990) 1791–1797.
21 P.J. Gemperline, J.R. Long and V.G. Gregoriou, Nonlinear multivariate calibration using principal components regression and artificial neural networks, *Analytical Chemistry*, 63 (1991) 2313–2323.
22 T. Aoyama, Y. Suzuki and H. Ichikawa, Neural networks applied to quantitative structure–activity relationship analysis, *Journal of Medicinal Chemistry*, 33 (1990) 2583–2590.
23 T. Aoyama, Y. Suzuki and H. Ichikawa, Obtaining the correlation indices between drug activity and structural parameters using a neural network, *Chemical Pharmaceutical Bulletin*, 39 (1991) 372–378.
24 T. Aoyama and H. Ichikawa, Basic operating characteristics of neural networks when applied to structure–activity studies, *Chemical Pharmaceutical Bulletin*, 39 (1991) 358–366.
25 C. Borggaard and H.H. Thodberg, Optimal minimal neural interpretation of spectra, *Analytical Chemistry*, 64 (1992) 545–551.
26 I.E. Alguindigue and R.E. Uhrig, Compression of spectral signatures using recirculation networks, *Scientific Computing and Automation*, May (1991) 43–50.
27 P. de B. Harrington, Fuzzy multivariate rule-building expert systems: minimal neural networks, *Journal of Chemometrics*, 5 (1991) 467–486.
28 J. Zupan and J. Gasteiger, Neural networks: a new method for solving chemical problems or just a passing phase?, *Analytica Chimica Acta*, 248 (1991) 1–30.
29 D.E. Rumelhart and J.L. McClelland, *Parallel Distributed Processing*, Vol. 1, MIT Press, Cambridge, MA, 1986.
30 P.J. Werbos, Beyond regression: new tools for prediction and analysis in the behavioral sciences, *Doctoral Dissertation*, 1974.
31 E.D. Sontag, Remarks on interpolation and recognition using neural nets, in R.P. Lippmann, J.E. Moody and D.S. Touretzky (Editor), *Advances in Neural Information Processing Systems*, Vol. III, Morgan Kaufmann, San Mateo, CA, 1991.
32 E.D. Sontag, Feedforward nets for interpolation and classification, *Journal of Computer and Systems Science*, in press.
33 R. Hecht-Nielsen, *Neurocomputing*, Addison-Wesley, New York, 1991.
34 D. Tveter, Getting a fast break with backprop, *AI Expert*, (1991) 36–43.
35 R.A. Jacobs, Increased rates of convergence through learning rate adaptation, *Neural Networks*, 1 (1988) 295–307.
36 T. Tollenaere, SuperSAB: Fast adaptive backpropagation with good scaling properties, *Neural Networks*, 3 (1990) 561–573.
37 J. Hertz, A. Krogh and R.G. Palmer, *Introduction to Theory of Neural Computation*, Addison-Wesley, New York, 1991.
38 S.N. Deming and S.L. Morgan, Simplex optimization of variables in analytical chemistry, *Analytical Chemistry*, 45 (1973) 278–283A.
39 S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, Optimization by simulated annealing, *Science*, 220 (1983) 671–680.
40 L. Xu and E. Oja, Improved simulated annealing, Boltzmann machine and attributed graph matching, in L.B. Almeida and C.J. Wellekens (Editors), *Neural Networks: EURASIP Workshop 1990, Semsimbra, Portugal, February 1990, Proceedings*, Springer-Verlag, New York, 1990.
41 I.O. Bohachevsky, M.E. Johnson and M.L. Stein, Generalized simulated annealing for function optimization, *Technometrics*, 28 (1986) 209–217.
42 Y. Chauvin, Generalization performance of overtrained backpropagation networks, in L.B. Almeida and C.J. Wellekens (Editors), *Neural Networks: EURASIP Workshop 1990, Semsimbra, Portugal, February 1990, Proceedings*, Springer-Verlag, New York, 1990.
43 A.S. Weigend, D.E. Rumelhart and B.A. Huberman, Backpropagation, weight elimination, and time series prediction, in D.S. Touretzky, J.L. Elman, T.J. Sejnowski and G.E. Hinton (Editors), *Connectionist Models, Proceedings of the 1990 Summary School*, Morgan Kaufmann, San Mateo, CA, 1991.
44 Y. Chauvin, A backpropagation algorithm with optimal use of hidden units, in D.S. Touretzky (Editor), *Advances in Neural Information Processing Systems*, Vol. I, Morgan Kaufmann, San Mateo, CA, 1989.