NEURAL NETWORKS IN CIVIL ENGINEERING. I: PRINCIPLES AND UNDERSTANDING

By Ian Flood,[1] and Nabil Kartam,[2] Associate Members, ASCE

ABSTRACT: This is the first of two papers providing a discourse on the understanding, usage, and potential for application of artificial neural networks within civil engineering. The present paper develops an understanding of how these devices operate and explains the main issues concerning their use. A simple structural-analysis problem is solved using the most popular form of neural-networking system—a feedforward network trained using a supervised scheme. A graphical interpretation of the way in which neural networks operate is first presented. This is followed by discussions of the primary concepts and issues concerning their use, including factors affecting their ability to learn and generalize, the selection of an appropriate set of training patterns, theoretical limitations of alternative network configurations, and network validation. The second paper demonstrates the ways in which different types of civil engineering problems can be tackled using neural networks. The objective of the two papers is to ensure the successful development and application of this technology to civil engineering problems.

INTRODUCTION

In recent years, there has been a growing interest in a class of computing devices that operate in a manner analogous to that of biological nervous systems. These devices, known as artificial neural networks, or connectionist systems, are finding applications in almost all branches of science and engineering. Applications in civil engineering only go back to the late 1980s (Flood 1989), but already cover a range of topics as diverse as process optimization (Flood 1990a; Garrett et al. 1993), determining the loads on the axles of fast-moving trucks (Gagarine et al. 1992), construction simulation (Flood 1990b), seismic hazard prediction (Wong et al. 1992), classification of nondestructive evaluation signals (Garrett et al. 1993; Upda and Upda 1991), estimating construction costs (Moselhi 1991), and the selection of vertical formwork systems (Kamarthi et al. 1992). Several factors have stimulated this interest, the most notable being the recognition of the promise of certain information-processing characteristics apparent in the brain that have eluded capture within the conventional electronic digital-computing environment. These include the ability to learn and generalize from examples, to produce meaningful solutions to problems even when input data contain errors or are incomplete, to adapt solutions over time to compensate for changing circumstances, to process information rapidly, and to transfer readily between computing systems. Other factors that have spurred interest include advances in artificial neural networks [such as the development of the generalized delta rule (Rumelhart et al. 1986)] and the advent of affordable desktop-computing systems sufficiently powerful to handle the large computational loads imposed by artificial-neural-network training algorithms.

[1]Asst. Prof., Dept. of Civ. Engrg., Univ. of Maryland, College Park, MD 20742.
[2]Asst. Prof., Dept. of Civ. Engrg., Univ. of Maryland, College Park, MD.

Note. Discussion open until September 1, 1994. Separate discussions should be submitted for the individual papers in this symposium. To extend the closing date one month, a written request must be filed with the ASCE Manager of Journals. The manuscript for this paper was submitted for review and possible publication on March 29, 1993. This paper is part of the ***Journal of Computing in Civil Engineering***, Vol. 8, No. 2, April, 1994. ©ASCE, ISSN 0887-3801/94/0002-0131/$2.00 + $.25 per page. Paper No. 5789.

Current artificial neural networks are much less complex than their biological counterparts, comprising many fewer components and operating in a manner that is greatly abstracted. For example, the number of neurons in what is currently considered a sizable network would be on the order of 1,000 neurons with 1,000,000 connections [see Gagarine et al. (1992), for example], whereas the human brain is believed to comprise as many as 10^{11} neurons with perhaps 10^{14} connections (Rumelhart et al. 1986) and in this sense is 100,000,000 times more complicated. The quest, however, at least as far as current engineering applications of artificial neural networks are concerned, is not to replicate the size and complexity of biological neural systems, but rather to exploit their essential information-processing characteristics, such as those of generalization and error tolerance. This can be accomplished in networks composed from relatively few neurons.

Neural networks should not be viewed so much as an alternative to conventional computing techniques, and procedural- and symbolic-processing methods but as a complement. Indeed, many researchers are considering the use of hybrid systems integrating artificial neural networks with, in particular, knowledge-based expert systems (Ferrada et al. 1990; Kartam et al. 1994) in order to exploit the advantages specific to each technique. Neither should artificial neural networks be considered a remedy for all existing computational failings. In fact, they have many shortcomings of their own, most notably, the production of inexact solutions, a lack of theory to guide selection of the most appropriate size and configuration for a network, and slow progress during training. Such issues need to be resolved before the potential of artificial neural networks can be realized, although significant progress is being made in many of these areas (Flood 1993).

The present paper is the first of two papers that introduce the use of neural networks in civil engineering. Together, the papers are designed to help researchers to identify instances when this new tool is applicable in solving a given civil engineering problem, and to provide guidance in the initiation and execution of that research. To this end, the papers include a number of tips on the successful implementation of neural networks.

There is a tendency among users to throw a problem blindly at a neural network in the hope that it will formulate an acceptable solution, and this is arguably necessary due to the lack of neural-network theory. This habit is perhaps encouraged by the ability of neural networks to develop a solution to a problem automatically. This paper aims to ensure that decisions about the design of a neural-network implementation are, as far as possible, made in a purposeful manner and with understanding of their consequences. For this reason, a graphical interpretation of the operation of neural networks is included, providing insight into how and why these devices work. The basis of this presentation is a simple structural-analysis problem solved using the most popular form of neural-networking system, that is, a layered feed-forward network comprising synchronously operating neurons and trained in a supervised manner. The primary concepts and issues concerning the use of neural networks (such as their theoretical limitations and factors affecting their ability to operate efficiently) are introduced, making reference, where appropriate, to the same structural-analysis example.

CONCEPTS AND ISSUES

Neural networks are configured from a number of parallel operating processors, termed neurons, cells, or units, connected into some circuit, such as that illustrated in Fig. 1(*a*). Individually, the neurons perform trivial

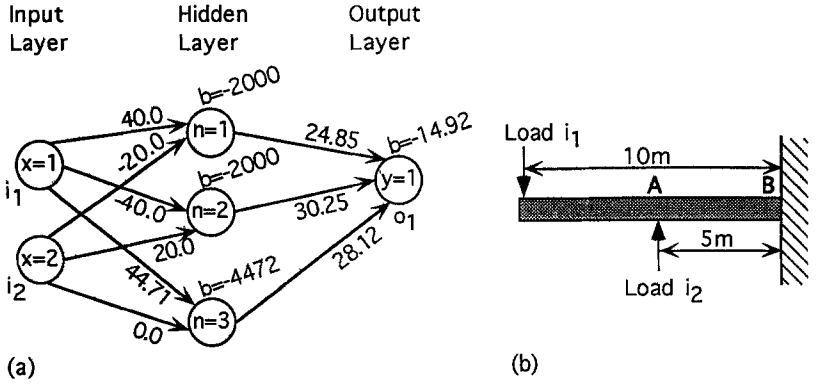

FIG. 1. Feedforward Network for Solving Structural Analysis Problem: (*a*) Six-Neuron Network; (*b*) Cantilever Loading

functions, but collectively, in the form of a network, they are capable of solving complicated problems. These problems range from the evaluation of production rates in construction processes (Karshenas and Feng 1992) to the prediction of tower-guy pretension (Issa et al. 1992).

There is a diverse range of neural networks in terms of structure and mode of operation. As an introduction, however, a neural network is developed to solve a simple structural-analysis problem, that of determining whether the loading on a cantilever will cause a critical bending moment to be exceeded. While this problem can be solved by more general alternative methods, it was selected since it is readily understood within the civil engineering profession and comprises a number of characteristics essential to a clear understanding of how neural networks work. The simple nature of this example should not be taken as an indication of the limits of neural networks—the ideas and concepts developed in this example extend directly to more complicated problems (comprising many dependent and independent variables) for which the neural-network approach is often the most appropriate solution available. Examples of such problems and their solving by use of the neural network approach are the subject of the second paper (Flood and Kartam 1994).

The type of network considered in this example falls into the most popular class, that of the layered feedforward network with synchronously operating neurons, and trained using a supervised scheme. Consideration, however, is also given to unsupervised training, specifically clustering algorithms, highlighting how it differs from supervised training.

Synchronous Feedforward Network Applied to Structural Analysis Problem

Fig. 1(*a*) illustrates a six-neuron network connected in a layered feedforward circuit. A problem is presented to the network as an array of real values, each element of which is entered to a different neuron in the input layer. The input neurons transmit these values across the links to the second (hidden) layer of neurons. On each link is a weight used to multiply transmitted values. The weighted values converging at a neuron in the hidden layer are summed along with a weighted bias associated with that neuron. The result is then put through a simple function to generate a level of activity for the neuron. The activation levels of the hidden neurons are then trans-

mitted across their outgoing links to the neurons in the output layer. As before, these values are weighted during transmission across the links, then summed at the output neuron and put through an activation function. The level of activity generated at the output neuron(s) is the network's solution to the problem presented at the inputs. All neurons within a layer in this type of network operate synchronously in the sense that, at any point in time, they will all be at the same stage in processing.

Typically, there will be many neurons at each layer in such a network, including the output layer, and often there will be more than one hidden layer. The following system of equations provide a generalized description of the mode of operation of this type of network, independent of the number of neurons in each layer:

$$h_n = f\left(\sum_{x=1}^{X} (w_{x,n} \cdot i_x) + b_n\right) \tag{1}$$

$$o_y = f\left(\sum_{n=1}^{N} (v_{n,y} \cdot h_n) + b_y\right) \tag{2}$$

where h_n = activity level generated at the nth hidden neuron; o_y = activity level generated at the yth output neuron; $w_{x,n}$ and $v_{n,y}$ = weights on the connections to the hidden and output layers of neurons, respectively; b_n and b_y = weighted biases; and $f[\ ]$ = activation function, in this case, the sigmoid function

$$f(t) = \frac{1}{1 + e^{-t}} \tag{3}$$

The task performed by a network is determined by the type of activation function adopted, the topology of the connections, and the values of the connection weights. Usually, the activation function and topology of connections are selected first and so it is left to determine an appropriate set of weights that make the network perform the required task. In the preceding example, the network assesses whether the loads on the cantilever shown in Fig. 1(*b*) cause the maximum bending moment to exceed 500 kNm. The loads, i_1 and i_2 (which operate exclusively in the directions indicated by the arrows), are entered to the corresponding input neurons in the network. If the level of activity generated at the output neuron is close to 0.0, the network has concluded that the loads will not produce a bending moment that exceeds 500 kNm. On the other hand, if the value output is close to 1.0, the conclusion is that the critical bending moment will be exceeded.

Training

A major concern in the development of a neural network is determining an appropriate set of weights that make it perform the desired function. There are many ways that this can be done; the most popular class of these algorithms are based on supervised training. Typically, supervised training starts with a network comprising an arbitrary number of hidden neurons, a fixed topology of connections, and randomly selected values for the weights. The network is then presented with a set of training patterns, each comprising an example of the problem to be solved (the inputs) and its corresponding solution (the targeted outputs). For the bending-moment problem the network was trained using 21 such patterns, a sample of which are

presented in Table 1. Each problem is input into the network in turn, and the resultant output is compared to the targeted solution providing a measure of total error in the network for the set of training patterns. The weights are then adjusted by small amounts as prescribed by some rule [in this case the generalized delta rule (Rumelhart et al. 1986) was used] so that on the next occasion the example problems are presented to the network, the error is reduced, and the output is closer to that required. Typically, the process is repeated many times until the network is able to reproduce, to within a specified tolerance, the corresponding solutions to each of the example problems. Following these many examples, it is anticipated that the network is able to generalize what it has practiced—in essence, is able to learn—to provide accurate solutions to exmaples of the problem not used during training.

In the case of the generalized delta rule (Rumelhart et al. 1986), the first derivative of the total error with respect to a weight determines the extent to which that weight is adjusted. Thus, the more a weight appears, in this sense, to affect the total error, the more that weight is changed. The direction of the change is that which decreases the total error. Provided the size of the changes are relatively small, this approach approximates a steepest-error-gradient descent.

Graphical Interpretation

Understanding how neural networks solve problems can be achieved most readily by means of a graphical interpretation of their operation. Referring to the network of Fig. 1(*a*), first imagine that the levels of activity generated at each of the hidden neurons are plotted against a range of values at the inputs. This would result in the three surfaces shown in Fig. 2(*a–c*), one for each hidden neuron. Each plot takes the form of a sigmoidal-shaped step as would be the result for any neuron of the type described by (1) and (3). The sigmoidal shape of the surface results from the use of the sigmoid activation fucntion—if an alternative activation function had been used, such as a sine or Gaussian one, the shape of the surfaces would change accordingly. The other characteristics of a sigmoidal step are determined by the values of the various weights associated with the hidden neuron. For example, the direction in which a step rises is the direction of the vector of its input weights $\mathbf{w}$; its slope is proportional to the length of that vector $\|\mathbf{w}\|$ (in this example the length of the vectors are great and so the sigmoid steps are steep); and its displacement from the origin is the ratio of the weighted bias to the length of the weight vector $-b/\|\mathbf{w}\|$. Clearly, a sigmoidal step

TABLE 1. Sample of Seven Training Patterns for the Bending-Moment Problem

	Input Values		Output Values
Training pattern (1)	i_1 (2)	i_2 (3)	o_1 (4)
1	0.0	0.0	0.0
2	49.0	0.0	0.0
3	51.0	0.0	1.0
4	100.0	0.0	1.0
5	120.0	25.0	1.0
6	25.0	50.0	0.0
7	75.0	51.0	0.0

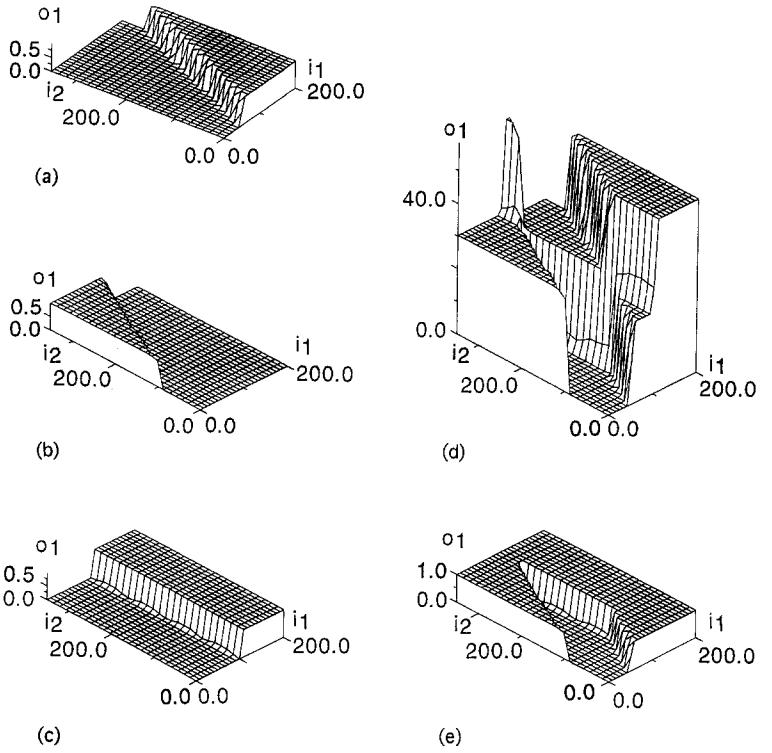

FIG. 2. Output From Neurons in Network of Fig. 1: (*a*) Hidden Neuron 1; (*b*) Hidden Neuron 2; (c) Hidden Neuron 3; (*d*) Summed Input to Output Neuron 1; (e) Final Result at Output Neuron 1

of any direction, slope, and displacement can be accomplished given an appropriate set of weights.

Similarly, the level of activity generated at the output neuron can be plotted against a range of values at the input neurons, producing the surface shwon in Fig. 2(*e*). This surface (representing the network's solution to a range of the bending-moment problem) is essentially a composite of the surfaces generated by the hidden neurons. The surfce generated at each hidden neuron is first multiplied by the weight on the link v_n, connecting it with the output neuron. This, in effect, changes its amplitude to the value of the weight v_n. The amplitude adjusted surfaces are then added together at the output to produce the surface shown in Fig. 2(*d*). This surface provides a rudimentary solution to the bending-moment problem if all values equal to and below 14.92 are taken to indicate a situation where the bending moment will not exceed the prescribed limit, and all other values are taken to mean that the limit will be exceeded. The tidier solution shown in Fig. 2(*e*) is achieved by first shifting the entire surface shown in Fig. 2(*d*) down by 14.92 (by subtracting the bias at the output neuron), and then pushing all areas above 0.0 up to about 1.0 and pushing all areas below 0.0 up to about 0.0 (by using the sigmoid activation function at the output neuron). This common form of neural network operates merely as a device for modeling surfaces (or hypersurfaces if there are more than two independent

variables in the problem), where solution surfaces are formed from an appropriate combination of primary surfaces, in this case, sigmoidal steps.

The graphical interpretation can be extended to any synchronous, layered feedforward network, irrespective of the number of neurons in each layer. This is made possible by three factors. First, an increase in the number of output neurons results in an increase in the number of solution surfaces generated by the network—a separate surface is produced at each output neuron. These may all be very different in form despite the fact that they are composed from the same set of primary surfaces. The differences between these solution surfaces can be attributed entirely to the differences in the output connection weights v_y. Second, an increase in the number of hidden neurons increases the number of sigmoidal steps available for combination at the output neurons, thus providing greater flexibility for modeling more convoluted solution surfaces. Finally, increasing the number of input neurons increases the number of spatial dimensions within which the surfaces operate (the number of dimensions equals the number of inputs plus one), and although it is not possible to plot these if there are more than three dimensions, the principles discussed remain the same.

It is possible to extend the graphical interpretation to networks comprising more than one hidden layer [see, for exmaple, Lapedes and Farber (1988)] and to networks with connection schemes other than that of the layered feedforward circuit, since these can always be converted to a feedforward equivalent [Minsky and Papert (1969)].

Generalization

In the graphical interpretation of supervised training, each training pattern is viewed as a point in space, located by plotting its output component against its input component. The training patterns in Table 1, for example, mark the points indicated in Fig. 3. Collectively, the points plotted in this manner infer a solution surface. The objective is to train the network to provide an acceptable approximation to this implied (targeted) surface.

The intention, however, is not simply to train a network to reproduce the solutions to the examples in the training set, but rather to find a generalized solution applicable to all examples of the problem that could be of interest. Fig. 4 shows three alternative solutions (represented as curves) to a problem where all that is known is a set of training patterns, a number of which are withheld from training for testing of the network. The sets of training and test patterns are the same in each graph. The first solution fits the training points exactly, though it fails to provide a generalized solution valid for the test problems. A more acceptable solution may be the second curve in the figure, which provides a closer approximation to the test patterns while still making an exact fit to the training points.

It is possible, however, that the values that make up the training patterns contain errors (perhaps due to inaccuracies in measurements or because the targeted solution changes with time), in which case, the second solution may be considered too literal an interpretation of the training points. If this is the case, a curve that follows a more general trend implied by the training patterns is more meaningful, such as the curve in the third solution in Fig. 4. Ensuring that a neural network provides the most appropriate degree of generalization as such is an outstanding issue, but it is dependent on the number and configuration of the hidden neurons, the number and dispersion of the training patterns, as well as on the type of network and training algorithm adopted. For example, if too few hidden neurons are included,

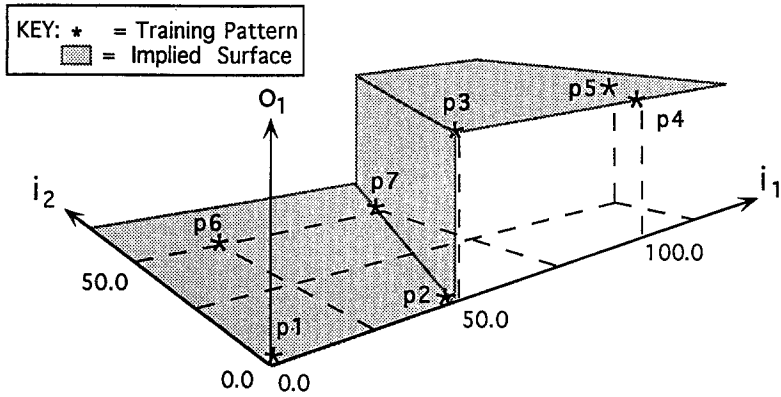

FIG. 3. Points Marked by Sample of Seven Training Patterns

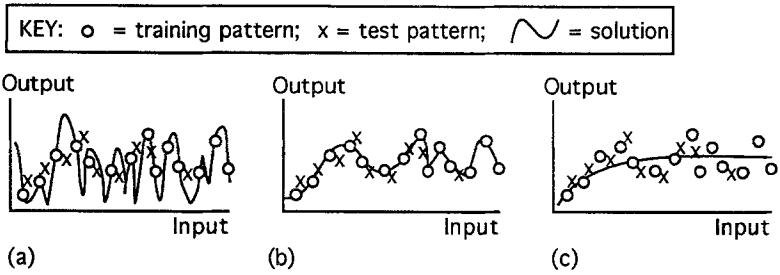

FIG. 4. Alternative Generalizations of Training-Pattern Set: (*a*) Poor Interpolation; (*b*) Literal Interpretation; (*c*) Following Trend

then the network is not able to model all the subtle changes in the shape of the solution surface, and a solution more like that shown in Fig. 4(*c*) is obtained. On the other hand, if too many hidden neurons are included, the solution surface produced by the network may develop false surface features (such as bumps or depressions) at points between the training patterns, as in Fig. 4(*a*). Similarly, a limited set of training patterns can provide insufficient information about localized features in the implied solution surface, making it impossible to train the network to produce accurate solutions to problems between the training points. The issues of selecting an appropriate set of training patterns and configuration for the hidden neurons are discussed in the following subsections.

Generalization as discussed so far has been concerned with interpolation. An alternative and equally important form of generalization, however, is extrapolation. Extrapolation occurs when a network is presented with a problem that falls outside the training domain—the region in the input space that embraces all patterns in the training set. Fig. 5 shows the training domain as defined by the 21 training patterns used for the bending-moment problem (including the seven in Table 1). Typically, neural networks are unable to produce accurate solutions far outside the training domain, since there is no information provided on the form of the solution surface in this region. An example of an exception to this rule is the network developed for the bending-moment problem that is able to provide correct solutions

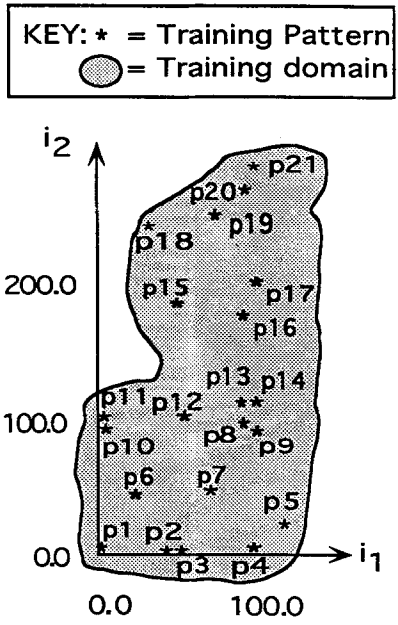

FIG. 5. Example Training Domain

for an unbounded range of problems in the positive i_1 and i_2 region, far beyond the training domain. This fortuitous exception is made possible by the fact that the cross section of the solution surface does not vary outside the training domain.

Graphical Interpretation Applied to Unsupervised Training

So far the analysis has focused on supervised training of networks. An alternative, but equally important approach to network development, is that of unsupervised training. In this case, the training patterns do not include solutions (targeted outputs), rather, the network is left to determine these automatically in the training process.

Typical of this category of training schemes are the clustering algorithms, such as those proposed by Kohonen (1984) for problem classification. In its simplest form, training proceeds by adjusting weights in a way that forces each output neuron to respond to as many training patterns as possible without overlapping with the response of other output neurons. The result is a division of the input space into regions containing distinct clusters of training patterns, each cluster embraced by a different output neuron. The class into which a problem is placed by the network is denoted by the output neuron that becomes most active.

In many clustering algorithms, the network is structured so that the output neurons enhance their own responses and suppress those of others. The result is that the output neuron with the greatest response tends toward its maximum possible level of activity, usually 1.0, while the others tend toward their lowest possible level of activity, 0.0. Thus, the solution surface generated at an output neuron forms a single high plateau, such as that illustrated in Fig. 6. The plateau is focused over the section of input space containing the cluster of training patterns classified by the output neuron.

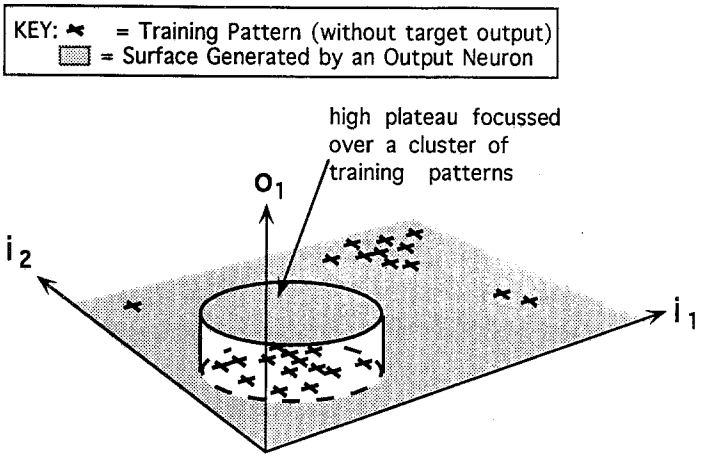

FIG. 6. Solution Surface for Output from Network Trained with Clustering Algorithm

TABLE 2. Bending Constraints Implemented by Hidden Neurons in Fig. 1

Bending moment constraint inequalities (1)	Hidden neuron implementing constraint (2)
Bending at A: $5i_1 \leqq 500$ (kNm)	Number 3
Bending at B: $10i_1 - 5i_2 \leqq 500$ (kNm)	Number 1
Bending at B: $10i_1 - 5i_2 \geqq -500$ (kNm)	Number 2

The bending-moment problem considered here is in essence one of classification—a load combination is categorized as either causing or not causing the maximum bending moment to be exceeded. In this case, however, the use of an unsupervised clustering algorithm is not appropriate, since it is unlikely that the training patterns form discernible clusters in the input plane that correspond to the two classes of loading. Problems of this sort therefore require some guide in training, other than pattern clustering, that enable the network to develop an appropriate classification. In the case of supervised training, the guide is the set of target outputs provided as part of the training patterns.

Number of Hidden Layers

For the bending-moment problem, a single hidden layer comprising three neurons is all that is required to construct the solution surface. In this case, the sigmoidal step generated at each hidden neuron implements one of the bending-moment constraint inequalities given in Table 2. Normally, however, the shape of the solution surface is more convoluted than that of Fig. 2(*e*), and therefore is not so conducive to construction out of sigmoidal-shaped steps. Moreover, it is likely that the shape is unknown or exists in a space with more than three dimensions, making it impossible to visualize. In all cases, it is not obvious how many hidden layers and neurons of each to include in the network. Determining an appropriate configuration of hidden neurons for a given problem is usually troublesome.

Hecht-Nielsen (1989) provides a proof that one hidden layer of neurons

(operating sigmoidal activation functions) is sufficient to model any solution surface of practical interest, though his line of reasoning does not reflect the way in which existing training algorithms operate, and in this sense is of little more than academic interest. An alternative, more pragmatic proof that two hidden layers are sufficient is provided by Lapedes and Farber (1988). In this case, it is shown that an appropriate combination of the sigmoidal-shaped steps generated at neurons in the first hidden layer can be combined to produce a bump-shaped feature (of any width and with its peak located at any position) at a neuron in the second hidden layer. It is then argued that any solution surface can be approximated by combining a suitable set of these bumps at an output neuron. It should be noted that both proofs are for networks that do not include sigmoid activation functions at the output neurons—the sigmoid function [as defined in (3)] can only generate values between 0.0 and 1.0 and thus, if used as an activation function at the output neurons, causes all points on the solution surfaces to fall between these values.

Despite Hecht-Nielsen's (1989) proof, there are many solution surfaces that are extremely difficult to model using a sigmoidal network comprising one hidden layer. Certainly, two hidden layers provide the greater flexibility necessary to model complex-shaped solution surfaces, and are thus recommended as a starting point when developing a layered feedforward network of sigmoidal neurons. Alternative configurations, with either just one or several hidden layers, should also be considered, since they often lead to more effective solutions. There are two notable exceptions to the two-hidden-layer rule: first, any network comprising just one input neuron; and second, any problem where the input data assume binary values (typically 0 and 1, though the same is true for other pairs of values). In both cases, it needs to be shown that a solution can be found readily using a single hidden layer, thus recommending such a configuration as a starting point.

Number of Hidden Neurons

Generally, there is no direct and precise way of determining the most appropriate number of neurons to include in each hidden layer, and this problem becomes more complicated as the number of hidden layers in the network increases. From the understanding developed in the graphical interpretation, it appears that increasing the number of hidden neurons provides a greater potential for developing a solution surface that fits closely to that implied by the training patterns. In practice, however, a large number of hidden neurons can lead to a solution surface that, while fitting the training points closely, deviates dramatically from the trend of the surface at intermediate points [as in Fig. 4(*a*)] or that provides too literal of an interpretation of the training points [as in Fig. 4(*b*)]. In addition, a large number of hidden neurons slow down operation of the network, both during training and in use, if it is implemented using a software emulator as is usually the case. Conversely, an accurate model of some or all features in the solution surface may not be achieved if too few hidden neurons are included in the network. In an attempt to resolve this dilemma, a range of different configurations of hidden neurons is normally considered, and that with the best performance is accepted. Use could be made, however, of a training system that evaluates automatically the utility of alternative configurations of hidden neurons. One such technique (Karnin 1990) starts by training a relatively large network that is later reduced in size by removing the hidden neurons that do not significantly contribute to the solution. Yet another approach

is the radial-Gaussian system developed by Flood (1991, 1993), which adds hidden neurons to the network in a sequential manner, training each on the error left over from its predecessors. The number of hidden neurons required to achieve a given level of accuracy is thus determined automatically during training. A description and application of this latter system is presented by Gagarin et al. (1994).

Number, Distribution, and Format of Training Patterns

Another factor that can significantly influence a network's ability to learn and generalize is the number of patterns in the training set. Increasing the number of training patterns provides more information about the shape of the solution surface(s), and thus increases the potential level of accuracy that can be achieved by the network. A large training-pattern set, however, can sometimes overwhelm certain training algorithms, thereby increasing the likelihood of an algorithm becoming stuck in a local error minimum. Consequently, there is no guarantee that adding more training patterns leads to improved solutions. Moreover, there is a limit to the amount of information that can be modeled by a network that comprises a fixed number of hidden neurons; thus, the gains that can be made by adding more training patterns quickly decrease.

There are a number of practical factors that may limit the number of patterns that can be used for training. For many problems, a fixed set of training patterns may be all that is available, and collecting or producing more may be impossible or prohibitively expensive. Furthermore, the time required to train a network increases with the number of patterns in the training set. Thus, even if it is possible to produce as many training patterns as required (such as is the case with the bending-moment problem, where an alternative model is available to produce them at little cost), the rate of progress of training may become unacceptably slow.

Generally, where there is some control over the collection of training patterns, pilot experiments should be performed to determine an appropriate size for the training set. The first experiment may use a relatively small set of training data (perhaps between 10 and 100 examples) well distributed within the problem domain. The performance of the network during training should then be monitored (as explained in the upcoming section on validation), and observations made of the level of accuracy on which the network appears to be converging and the approximate time required to approach this limit. The experiment should then be repeated using a larger set of training patterns (doubling or even increasing by the size of the set). A third experiment may also be performed using an intermediate number of training patterns. The results from these experiments provide the developer with a feel for the sensitivity of the performance of the network compared to the size of the training-pattern set. A better estimate of the number of training patterns required can then be made, though this choice should balance the accuracy of the results achieved against both the time required to train the network (if critical) and the cost of collecting or producing the training patterns.

The distribution of the training patterns within the problem domain (the region in the input space embracing all problems that could be input to the network) can have a significant effect on the learning and generalization performance of a network. Since artificial neural networks are not usually able to extrapolate, the training patterns should go at least to the edges of the problem domain in all dimensions. In other words, the problem domain

should be equal to, or a subset of, the training domain. It is also advisable to have the training patterns evenly distributed within this region. If this is not the case, training tends to focus on those regions where training patterns are densely clustered and neglect those that are sparsely populated. It can be beneficial, however, to have concentrations of training patterns in areas where the form of the solution surface is more complicated. For example, in the case of bending-moment problem, training is facilitated if there are concentrations of training patterns close to the ridges in the solution surface [see Fig. 2(*a*)].

Since their distribution depends on the manner in which they are produced or collected, it is, unfortunately, not always possible to control training patterns across the problem domain. There are many ways in which training patterns may be acquired, the most notable being

- Synthesis of historic data, an example of which is the use of published construction-equipment-performance data to train a network to estimate production in earthmoving operations (Karshenas and Feng 1992).
- Direct observation of the situation to be modeled, or of a similar situation [see, for example, Flood (1990b)].
- Consultation with a human expert, as in knowledge acquisition for expert systems [see, for example, Levitt and Kartam (1990)].
- Analysis of an alternative model of the situation under consideration, an example being the use of the constraint equations in Table 2 to set up training patterns for the bending-moment problem.
- Analysis of an inverse model of the situation under consideration. This approach to data acquisition was used in developing a network to estimate loading attributes of trucks from the strain response of the bridge over which they are traveling (Gagarine et al. 1992). A finite-element model that performed the inverse function (that of determining the strain response of a bridge given different types of trucks, truck loading, and velocities) was used to set up the training patterns.

It is clear that the first two methods of data collection provide little control over the distribution of training patterns over the problem domain, and total control is not necessarily available for the last three methods either.

The progress of training can be impaired if the training patterns mark a region that is relatively narrow in some dimensions and elongated in others, such as in Fig. 7(*a*). In these situations, changes in the shape of the solution surface during training may occur too rapidly in the directions in which the training domain is narrow or too slowly in the directions where it is broad. This problem can be alleviated by normalizing the training patterns across each input, thereby improving the proportions of the problem domain, as in Fig. 7(*b*).

Normalization can also be applied to the output values of training patterns so that the amplitude of the solution surface at each output is the same. This can be beneficial in situations where there is a large differential in the amplitudes of the solution surfaces targeted at each output. Otherwise training may tend to concentrate (at least in the earlier stages) on learning the targeted solution surface with the greatest amplitude.

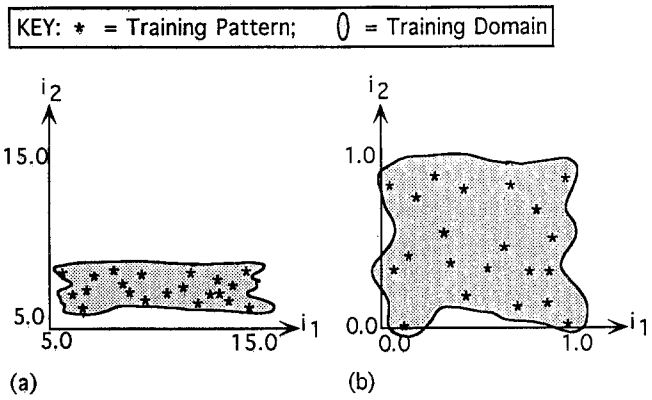

FIG. 7. Effect of Normalizing Input Training Data: (*a*) Training Domain Before Normalization; (*b*) Training Domain After Normalization

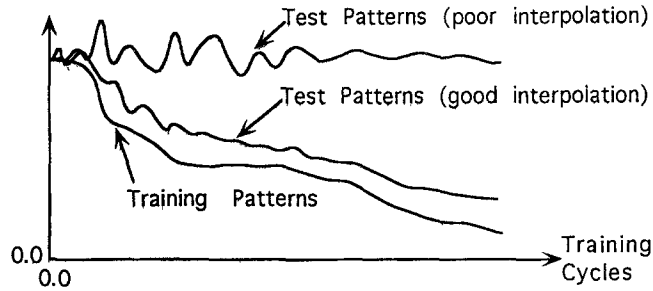

FIG. 8. Monitoring Network Performance

Validation of Network

Before a neural network can be used with any degree of confidence, there is a need to establish the validity of the results it generates. A network could provide almost perfect answers to the set of problems with which it was trained, but fail to produce meaningful answers to other examples, as in Fig. 4(*a*). Usually, validation involves evaluating the network performance on a set of test problems that were not used for training, but for which solutions are available for comparison. The correct solutions and those produced by the network may be compared in a qualitative manner (such as a visual comparison of plotted points) or in a quantitative manner using a statistical test (such as the correlation coefficient). The test problems should be selected in a manner that reduces the likelihood of significant bias in results. Usually, this is done by retaining a random sample of the training patterns, which is a satisfactory method provided the original set of training patterns are representative of the problems likely to be presented to the network. Evaluation of the network on the test problems should be undertaken after training has been completed. A more informative measure of the progress of training, however, is obtained by evaluating the validity of the network with these test problems at intervals during the course of training, as in Fig. 8. If the network is learning an accurate generalized solution to the problem, the average error curve for the test patterns decreases at a rate approaching that of the training patterns. At the other

extreme, if the network is merely learning the training patterns, the average error curve for the test patterns displays no downward trend.

There are many situations where solutions to example problems are not readily available [such as the optimization class of problems discussed in Flood and Kartam (1994)] and the task of validation is complicated. One way around this problem is to compare the performance of the neural network with an established alternative method of solving the problem. Alternatively, human conjecture could be used to evaluate the network's performance—this could be formalized within an expert system as proposed by Okagaki (1990).

Processing Speed

The rapid speed with which neural networks generate solutions to instances of a problem can be attributed to the fact that the task performed by each neuron is very simple and, in the case of feedforward circuits, that just one pass through the network is required. Currently, most networks are implemented using emulation software on a general-purpose digital computer. So although neurons are viewed as operating in parallel, the processing is actually performed serially. Dedicated neural-network hardware, however, is becoming available (Hecht-Nielsen 1990), and it exhibits the considerable advantage that the time required to produce a solution is not dependent on the number of neurons in each layer.

Processing speed can be a problem, however, when it comes to training the network. Fig. 9 shows that the speed of processing in this context is dependent on many factors. Typically, training proceeds until the measured error in the network is reduced to an acceptable level. For most problems, training must go through many cycles before this threshold of error is achieved. The number required is dependent on the reduction in error at each training cycle. This rate of convergence is, in turn, dependent on factors including complexity of the problem and the values adopted for the learning parameters [such as the learning rate and momentum factor in the generalized delta rule (Rumelhart et al. 1986)]. The amount of processing involved with each training cycle is another factor influencing the time required to train a network, in particular, this is influenced by the number of training patterns

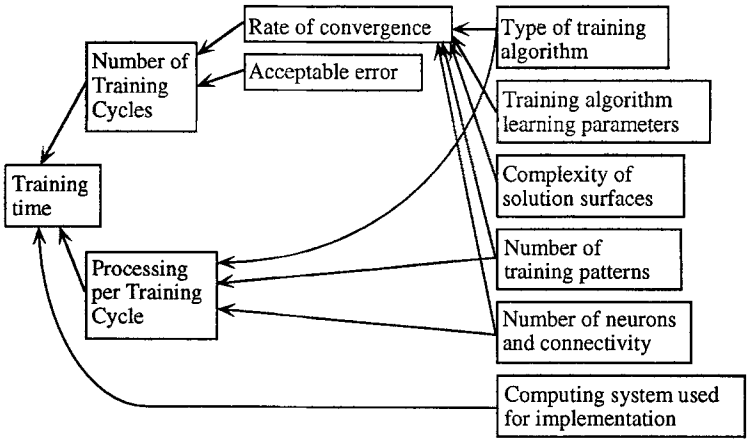

FIG. 9. Primary Factors Affecting Training Time

employed, the number of layers and (if using a serial implementation) the number of connections per layer. Consequently, there is an incentive to minimize as far as possible the number of neurons incorporated in the network and the number of patterns used for training.

There are, however, many developments available that expedite the progress of training a network. One such example that has been applied within civil engineering (Gagarine et al. 1992) is the radial-Gaussian networking system and training algorithm developed by Flood (1991, 1993). This technique provides training speeds that are much faster than that of the generalized delta rule and includes only as many neurons in the network as required to achieve a given level of accuracy (measured on either training or test patterns), thus avoiding unnecessary processing.

CONCLUSION

Neural networks offer several advantages over the more conventional procedural and symbolic approaches to computing. The most frequently cited of these are their ability to develop a generalized solution to a problem from a set of examples, and to continue development and adapt to changing circumstances with exposure to new variations of a problem. The attribute of generalization permits them to be applied to problems other than those used for training and to produce valid solutions even when there are errors in the training data or in the description of an instance of a problem requiring a solution. These fctors combine to make neural networks a powerful tool for modeling problems in which functional relationships between dependent and independent variables are poorly understood, subject to uncertainty, or likely to vary with the passage of time. Such problems are commonplace throughout the various disciplines of civil engineering.

Problems where the time required to generate solutions is critical, such as real-time applications that require many solutions in quick succession, mark another important area that can benefit from the neural-network approach. In particular, the ability of neural networks to produce solutions in a fraction of a second, irrespective of the complexity of the problem, makes them valuable even when alternative techniques are available that can produce more optimal or more accurate solutions. For example, many intelligent-search optimization algorithms operate extremely slowly when the problem to be solved comprises many variables. In such situations, significant time can be saved using a neural network to find a good initial solution that acts as the starting point for the optimization algorithm (Issa et al. 1992; Wei 1993). On other occasions, a neural network may be used as a quick check on the solution developed by a more time-consuming in-depth analysis.

The present paper has concentrated on the most popular class of neural-networking system (the layered feedforward network comprising synchronously operating neurons and trained in a supervised manner) and looked at how and why such a system works and the primary issues concerning its use. There are, however, many alternative forms of neural-networking systems and, indeed, many different ways in which they may be applied to a given problem. The success of an implementation is dependent not just on the factors discussed (such as ensuring that a network is used only for problems that fall within the training domain), but also on the selection of an appropriate paradigm and strategy for application. The suitability of a given selection is very much dependent on the type of problem to be solved. These points are discussed in Flood and Kartam (1994), which provides a

description of the distinctive features of alternative forms of networking systems and a practical guide on how they can be applied to different classes of civil engineering problems.

ACKNOWLEDGMENTS

The writers gratefully acknowledge the financial support received from NSF grant MSS-9210721, the University of Maryland Minta Martin awards, and the Maryland Industrial Partnerships. Many thanks to Tanit Tongthong for reviewing and commenting on the paper.

APPENDIX. REFERENCES

Ferrada, J., Osborne-Lee, I., and Cuccuzzella, N. (1990). "Expanding the power of expert systems applications with neural networks." *Summer Nat. Meeting of the Am. Inst. of Chemical Engrs.*, San Diego, Calif.

Flood, I. (1989). "A neural network approach to the sequencing of construction tasks." *Proc., 6th Int. Symp. on Automation and Robotics in Constr.*, Construction Industry Institute, Austin, Tex., 204–211.

Flood, I. (1990a). "Solving construction operational problems using artificial neural networks and simulated evolution." *Proc., CIB W-55 and W-65 Joint Symp. on Build. Economics and Constr. Mgmt.*, Vol. 6, University of Technology, Sydney, Australia, 197–208.

Flood, I. (1990b). "Simulating the construction process using neural networks." *Proc., 7th Int. Symp. on Automation and Robotics in Constr.*, ISARC, Bristol Polytechnic, Bristol, U.K.

Flood, I. (1991). "A Gaussian-based neural network architecture and complementary training algorithm." *Proc., Int. Joint Conf. on Neural Networks*, Vol. I, Institute of Electrical and Electronic Engineers, New York, N.Y., 171–176.

Flood, I. (1993). "Training and use of networks of radial-Gaussian neurons." Submitted to IEEE Transactions on Neural Networks, Institute of Electrical and Electronic Engineers, New York, N.Y.

Flood, I., and Kartam, N. (1994)."Neural networks in civil engineering. II: Systems and application." *J. Comp. in Civ. Engrg.*, ASCE, 8(2), 149–162.

Gagarine, N., Flood, I., and Albrecht, P. (1992). "Weighing trucks in motion using Gaussian-based neural networks." *Proc., Int. Joint Conf. on Neural Networks*, Vol. II, Institute of Electrical and Electronic Engineers, New York, N.Y., 484–489.

Gagarine, N., Flood, I., and Albrecht, P. (1994). "Computing truck attributes with artificial neural networks." *J. Comp. in Civ. Engrg.*, ASCE, 8(2), 179–200.

Garrett, J., Case, M., Westervelt, J., Hall, J., Yerramareddy, S., Herman, A., Sim, R., and Ranjithan, S., (1993). "Engineering applications of neural networks." *J. Intelligent Manufacturing*, 4(1).

Hecht-Nielsen, R. (1989). "Theory of the backpropogation neural network." *Proc., Int. Joint Conf. on Neural Networks*, IEEE, Washington, D.C., Vol. I, 593–605.

Hecht-Nielsen, R. (1990). *Neurocomputing*. Addison-Wesley Publishing Co., Reading, Mass.

Issa, R., Fletcher, D., and Cade, R. (1992). "Predicting tower guy pretension using a neural network." *Proc., 8th Conf. in Comp. in Civ. Engrg. and Geographic Information Systems*, ASCE, New York, N.Y., 1074–1081.

Kamarthi, S., Sanvido, V., and Kumara, R. (1992). "Neuroform—neural network system for vertical formwork selection." *J. Comp. in Civ. Engrg.*, ASCE, 6(2), 178–199.

Karnin, E. D. (1990). "A simple procedure for pruning back-propagation trained neural networks." *Trans. on Neural Networks*, Institute of Electrical and Electronic Engineers, New York, N.Y., 1(2), 239–242.

Karshenas, S., and Xin, F. (1992). "Application of neural networks in earthmoving equipment production estimating." *Proc., 8th Conf. in Comp. in Civil Engrg. and Geographic Information Systems*, ASCE, New York, N.Y., 841–847.

Kartam, N., Flood, I., and Tongthong, T. (1994). "Integrating knowledge-based systems and artificial neural networks for civil engineering." *J. Artificial Intelligence in Engrg. Des., Anal. and Manufacturing*, Academic Press.
Kohonen, T. (1984). *Self-organization and associative memory*. Springer-Verlag KG, Berlin, Germany.
Lapedes, A., and Farber, R. (1988). "How neural networks work." *Neural information processing systems*, American Institute of Physics, 442–456.
Levitt, R., and Kartam, N. (1990). "Expert systems in construction engineering and management: state of the art." *The Knowledge Engrg. Rev.*, 5(2), 97–125.
Minsky, M., and Papert, S. (1969). *Perceptrons: an introduction to computational geometry*. MIT Press, Cambridge, Mass.
Moselhi, O., Hegazy, T., and Fazio, P. (1991). "A hybrid neural network methodology for cost estimation." *Proc., 8th Int. Symp. on Automation and Robotics in Construction*, ISARC, Stuttgart, Germany.
Okagaki, K. (1992). "Using expert systems to validate and verify neural networks." *Int. Neural Network Conf.*, Vol. 1, Paris.
Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). "Learning internal representations by error propagation." *Parallel Distributed Processing*, Vol. 1, D. E. Rumelhart and J. McClelland, eds., MIT Press, Cambridge, Mass.
Udpa, L., and Udpa, S. S. (1991). "Neural networks for the classification of nondestructive evaluation signals." *IEE Proc., -F*, 138(1), 41–45.
Wei, C. H. (1993). "Priority programming for transportation networks using artificial neural networks," PhD thesis, University of Maryland, at College Park, Md.
Wong, F., Tung, A., and Dong, W. (1992). "Seismic hazard prediction using neural nets." *Proc., 10th World Conf. on Earthquake Engrg.*, 339–343.