NEURAL NETWORKS IN CIVIL ENGINEERING. II: SYSTEMS AND APPLICATION

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

ABSTRACT: The first paper of this two-paper series developed an understanding of how artificial neural networks work and explained the primary issues involved in their use. This second paper demonstrates the versatility of neural networks as a problem-solving tool, and shows how they can be applied to different problems in civil engineering. Initially, the basic characteristics of neural networks and the variety of systems available are identified. The significance of these characteristics in terms of solving different classes of problems is considered. A range of different types of civil engineering problems is examined (in particular, vector mapping, dynamic-systems modeling, problems in which objectives vary with time, and optimization problems) and approaches to their solutions using different neural network paradigms are demonstrated.

INTRODUCTION

This paper builds on the understanding developed in the first paper (Flood and Kartam 1994) by illustrating how different classes of problems arising in civil engineering can be tackled using alternative forms of neural-networking systems.

An overview is provided of the alternative types of neural-networking systems available. The objective is not to describe in detail the various paradigms, such as Boltzmann Machines (Hinton and Sejnowski 1986), Hopfield networks (Hopfield 1982; Hopfield and Tank 1985), and Kohonen Networks (Kohonen 1984) [for such a description see Caudill (1987) or Hecht-Nielsen (1990)]. Rather, the intention is to provide a more general view of the range of features available among the alternative systems, and their relevance for solving different types of problems. The paper then describes how different classes of problems can be approached by exploiting alternative types of neural-networking systems.

CHARACTERISTICS OF NEURAL-NETWORKING SYSTEMS

The range of alternative neural-networking systems is immense and growing rapidly. Each system has its own characteristics, often determining its suitability for solving certain classes of problems. A useful means of putting this range into perspective is to classify them according to four principal characteristics: the format and interpretation of data; the connectivity of the network; its mode of operation; and the mechanism by which it is trained or developed. Each of these characteristics are discussed in turn.

Form of Data and Interpretation

Data input or output from a network is usually either continuous or discrete, though sometimes it may be symbolic (nonnumeric in form) or a

[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 0887-3801/94/0002-0149/$2.00 + $.25 per page. Paper No. 5790.

mixture of all these types. For example, the inputs to the bending-moment network in the first paper (Flood and Kartam 1994) (loads i_1 and i_2) are real (continuous) in form, while the output is a binary (discrete) value. The binary output is interpreted as a judgment of whether or not the maximum permissible bending moment will be exceeded. In contrast, the outputs from the network system developed by Flood (1989, 1990a) for determining optimal sequences to the flow-shop problem are real in form. In this case, the relative values of the outputs are interpreted as the sequence in which jobs are to be processed. As a final example, the network for selecting vertical formwork systems developed by Kamarthi et al. (1992) receives a mixture of real and binary values at its inputs, and generates real-valued outputs. The value generated at each output is interpreted as a level of recommendation for use of a particular formwork system.

The method of representing and interpreting data in a given situation is determined primarily by the problem to be solved and the type of network adopted, though there is often some margin for choice. An input variable used to indicate one of four sizes ("small," "medium," "large," and "extra large") could, for example, be represented by: (1) A single real-valued input to the network with values 0.0, 0.33, 0.67, and 1.0; (2) two binary valued inputs with values 00, 01, 10, and 11; or (3) four binary valued inputs with values 0001, 0010, 0100, and 1000. A similar choice of formats may be available at the outputs from a network. Testing the various formats available is the only means of determining which is the most effective for a given application. The binary approaches to data representation, however, increase the number of input or output neurons in the network, and thus the number of connection weights. Arguably, these extra degrees of freedom could facilitate the training task, though the same effect could be achieved by increasing the number of hidden neurons in the network.

Normalization of data so that the range of values at each input and/or at each output is the same is another choice of data representation that might be available. This can provide a better balance in the rate of learning in the network with respect to each independent and dependent variable [see the section on number, distribution, and format of training patterns in (Flood and Kartam 1994)].

Connectivity of Network

The architecture of a network, in terms of the way in which the neurons are connected, can take on just about any form. The most popular connection scheme, due to its simplicity, is the layered feedforward network such as that used for the bending-moment problem in Flood and Kartam (1994). In contrast, some systems use links in the network that connect backwards, allowing reprocessing of information. This is particularly useful for modeling recursive functions where successive states in a system are dependent on earlier states. An example as such is the neural-network system proposed for simulating construction processes (Flood 1990b) outlined in the later subsection on modeling dynamic processes.

If a problem is particularly complicated, it may be broken down into a number of parts, each of which is then solved separately by its own network module. More often than not, the choice of connection scheme for a network is arbitrary. For some systems, however, the connection scheme is defined precisely by the problem under consideration [such is the case for Hopfield networks (Hopfield and Tank 1985)]. In contrast, some networks undergo a change in connectivity during training [see for example Flood (1991)].

Mode of Operation of Neurons and Their Connections

Most networks comprise neurons that operate by integrating all incoming signals and placing the result through an activation function, often the sigmoid function. Values passing along links are multiplied by weights, and all components in the network operate both synchronously and in parallel. None of these conditions, however, are mandatory in a network; systems exist where signals arriving at a neuron are multiplied together rather than summed (Rumelhart et al. 1986), where values passing along links are processed by nonlinear functions (Pao 1989), and where neurons operate asynchronously (Hopfield 1982).

The radial-Gaussian networking system developed by Flood (1991) is an example of a system that deviates from the conventional mode of operation that has been applied successfully to a number of civil engineering problems [see, for example, Gagarine et al, (1994)]. In the radial-Gaussian system, values passed along a link emanating from the input layer are subtracted from an offset, and the result is squared; and the activation function used at the hidden neurons is Gaussian in form.

Method of Training the Network

It is meaningful to classify training algorithms, in terms of their suitability for solving different types of problems, into supervised and unsupervised schemes. In the former class, solutions to example problems are provided, which the network is supposed to learn or develop a generalized representation of, through the training process. Example algorithms include the generalized delta rule (Rumelhart et al. 1986) and the incremental technique for training radial-Gaussian networks (Flood 1991). The second class, that of unsupervised training schemes, are characterized by the absence of solutions to example problems. This may be because appropriate solutions are not available or because there is a desire to let the system identify by itself underlying patterns or groupings in a set of data. An example of unsupervised training applied to civil engineering is that of simulated evolution, which has been used to train a network to provide near optimal solutions to the flow-shop sequencing problem (Flood 1989, 1990a).

Supervised and unsupervised training schemes may operate both prior to and during application of the network. Training during application is suitable for problems that are continuously changing or in which there is room for significant improvement in performance as more information becomes available about the function being modeled. Training schemes may function by simply adjusting weights [as in the case of the generalized delta rule (Rumelhart et al. 1986)] or by adjusting the structure of the network [such as by adding neurons in stages (Flood 1991)].

APPROACHES TO DIFFERENT CLASSES OF PROBLEMS

There is no single way of tackling a given problem using neural networks. Care should be exercised when designing a neural network system to ensure that its specific characteristics, as outlined in the previous section, are suited to the problem at hand. In the following, appropriate methods of solving different classes of problems arising in civil engineering are illustrated. The simplest type of application of neural networks, that is as direct-mapping devices, is considered first. Methods of tackling larger problems are then examined, including, the use of a modular approach to network construction that facilitates decomposition of a problem into smaller, more manageable parts. The present paper then graduates to problems that are dynamic in nature, and thus require the use of recursive-, rather than direct-, mapping techniques. Finally, approaches to the solution of the special classes of transitory and optimization problems are discussed.

Mapping Problems

The network implementation for the bending-moment problem considered in the first paper (Flood and Kartam 1994) represents one of the most straightforward and common methods of applying neural networks—that is, it provides a direct mapping from a vector of inputs (representing the loads on the cantilever) to a vector of outputs (in this case a single value representing an assessment of whether the critical bending moment would be exceeded). This approach is applicable to problems were the solution can be described by a single vector comprising a fixed number of elements. Other examples in civil engineering of the use of neural networks as direct-mapping devices include the following:

- A system for selecting vertical formwork (Kamarthi et al. 1992). In this case, the network maps a set of inputs describing the situation in which a formwork system will operate onto a set of outputs representing recommendation levels for different formwork systems.
- Seismic hazard prediction (Wong et al. 1992). In this example, a network was trained to map a vector of inputs describing an earthquake event, local geology, and location data onto an output providing a forecast of its intensity at the position denoted.
- Predicting tower-guy pretension (Issa et al. 1992). This network maps from a vector describing key aspects of a guyed tower, to an output that estimates the optimal pretension for the cables.

An important subcategory of mapping networks are the classifiers. These devices classify problems into one of a finite set of solutions. There are two basic ways in which this can be undertaken. First, each output neuron may be limited to producing a finite set of values. In this case, the solution surfaces will in effect be segmented into a number of plateaus. The network used for the bending-moment problem in Flood and Kartam (1994) is a simple example of this type of classifier. The second approach is to associate a different class with each output neuron, and to make the classification on the basis of that which generates the greatest value. An example of this technique is the first-level network in the system developed by Gagarine et al. (1992) used to determine the class of truck crossing a bridge from the strain response of the girders.

Often, a problem is too cumbersome for solution by a network in the manner described, possibly requiring an unwieldy number of training patterns and neurons, thus making training impossible or at least unacceptably slow. One solution to this is to break a problem down into a number of clearly defined subproblems and to solve each of these with a different network. These network modules can be linked both in parallel and series, and may share the same inputs and/or use the output from other modules in the system as indicated in Fig. 1. A secondary advantage of this modular approach is that it can increase the scope of application of a system, enabling a user to construct a network for the specific task at hand from a set of pretrained modules. Furthermore, a user can extend the system to include

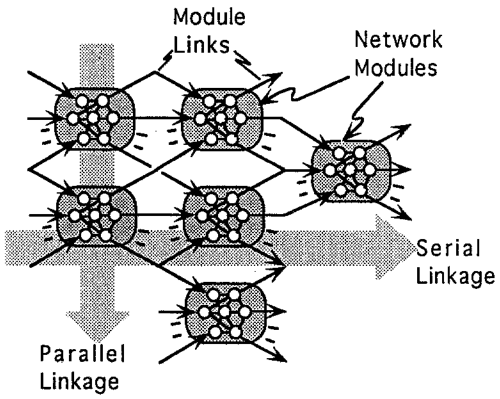

FIG. 1. Modular Network Structure

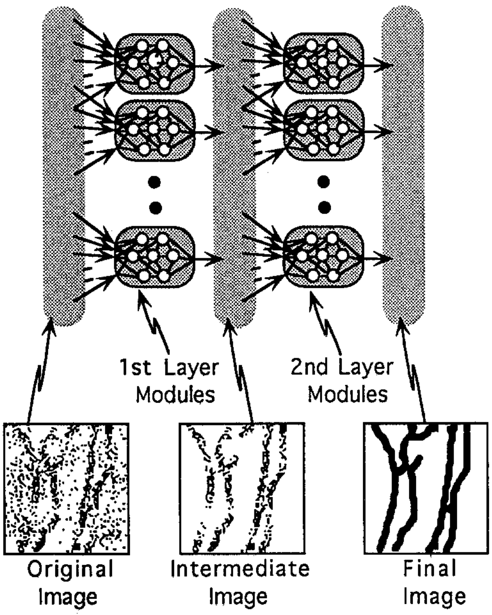

FIG. 2. Modular Network for Flaw Detection in Materials

new modules to suit changing needs without the burdensome task of having to redevelop the entire package from scratch.

A modular approach has been adopted in a system for enhancing images of flaws in the internal structure of construction components this system is currently under development at the University of Maryland's Department of Civil Engineering. The system is shown in Fig. 2. In this case, the network is divided into two layers of network modules, where the layers are linked in series. Each layer comprises a number of parallel-operating network modules.

The first layer is designed to process a scanned image of the internal structure of a component (this may be produced using X-ray photography or sonic-imaging techniques, for example) to filter out noisy information and irrelevant details such as those caused by joints, reinforcing bars, small voids, and minor density changes. The output from this layer is a cleaner image of those features that appear to represent flaws in the material. This first layer comprises a large number of network modules positioned in parallel. Each module has a number of inputs, each of which receives information about the level of gray at a point on the image. All the inputs for a given module receive these data from a semilocalized area of the image. The samples taken by the modules are allowed to overlap. A network module decides based on the information it receives, whether the part of the image it has sampled represents a flaw—the output from the module is a value representing its degree of certainty about the presence of a flaw. Since the features that determine the presence of a flaw are the same across the entire image, the modules in this layer have an identical set of weights.

The output from the first layer in the network is fed into a second, which again comprises many parallel network modules, this time designed to fill in missing features in the image of a flaw. The final result is a high-clarity image of the significant flaws in the component under inspection.

Only one module in each layer need be trained, since the module can be reproduced for the entire layer. Training is undertaken using a supervised scheme, which uses example patterns extracted from images of flawed components. A sample is taken at random from an image of the internal structure of a component and is presented to an expert at the computer screen, who is then asked to decide whether or not it represents a flaw. This procedure is repeated many times using many different images, until a comprehensive set of training patterns has been collected.

A primary advantage of the neural network approach to flaw detection, compared to the conventional algorithmic techniques, is that computation time is independent of the complexity of the problem (in this case the size and resolution of the image to be processed, and the amount of information required to differentiate between cracks and other image features) (Udpa and Udpa 1991). In addition, the technique is ideally suited to real-time applications, such as image-processing techniques, and it is faster and provides greater flexibility in terms of what can be removed and added to the image—this control is afforded by the choice of training patterns.

Often, the image needs to be passed through each layer in the network several times in order to ensure that all noisy features are removed and all missing features are completed. An example of this extension in the connectivity of a network (that is, allowing for feedback of information) is provided in the next subsection.

Other examples in civil engineering of the use of a modular approach to neural network development include:

- Simulation of construction activity (Flood 1990b). In this case, a network used to simulate a construction process is constructed from a number of modules (each representing a subprocess). The construct can be used to model a wide range of different processes by linking the modules as required. In addition, users can easily extend the construct by developing new modules suited to their particular needs.
- Estimating truck attributes from the strain response of the structure over which they are traveling (Gagarine et al. 1992). In this case, the network receives as input a vector of values representing the strain

measured at a fixed point on a bridge girder during the passage of a truck. Each element in the vector represents strain at a different point in time. Given this information, the network produces at its outputs a prediction of the velocity of the truck, the spacings of its axles, and the load on each axle. The network is arranged in two layers of modules. The first layer comprises a single network that determines the basic class of the truck. The second layer comprises separate modules, each estimating either the axle loads, axle spacings, or velocity of a truck of a specific class. The specific modules used in the second layer are determined by the class of the truck (determined by the first-layer network). Modularization was used to facilitate training of the system to an acceptable degree of accuracy within a reasonable period of time.

- Estimating earthmoving equipment production (Karshenas and Feng 1992). In this example, the network comprises a number of modules arranged in parallel, each dedicated to predicting the speed of an item of earthmoving equipment given certain environmental factors. The objective behind the modular approach in this case is to facilitate the inclusion and removal of new and obsolete equipment considered by the network.

Modeling Dynamic Processes

The types of problems considered so far have been largely limited to those that require the output of a single vector of results. Many problems, however, require the output of a series of results over time. An example of such is the simulation of construction processes where a prediction is required of the likely behavior and performance of a system at successive points in time. The advantages that accrue from using neural networks for dynamic modeling [rather than a conventional serial algorithm such as that described by Pilcher and Flood (1984) or Halpin and Woodhead (1976)] are facilitation of the development of the simulation model and increased speed of execution of a simulation run.

A basic strategy for applying neural computing to construction simulation, currently under investigation at the University of Maryland's Department of Civil Engineering, is to assemble a single network for each simulation study from a number of network modules that have been pretrained. Each of these modules are designed to function as some primary component in a model of a construction process such as a storage facility, a fleet of dump trucks, or a climatic conditions generator (see Fig. 3). Inputs to a module fall into two basic categories: (1) Those that are fixed during the course of the simulation representing, perhaps, in the case of a dump-truck cycle, the number of trucks, their loading capacities, and the travel distances; and (2) those that are variable, representing, for example, weather conditions and the availability of material—these inputs can be the output from other modules in the network. Once a model has been designed by a user, the necessary modules are linked into a complete network description and downloaded to the neural computer ready for simulation.

The completed network is initialized by inputting values that correspond to the state of the model at the start of simulation time. The network responds to this by producing as output the state of the model corresponding to the next point in simulation time (see Fig. 3). This output is transmitted back as input, thereby triggering another response in the network and,

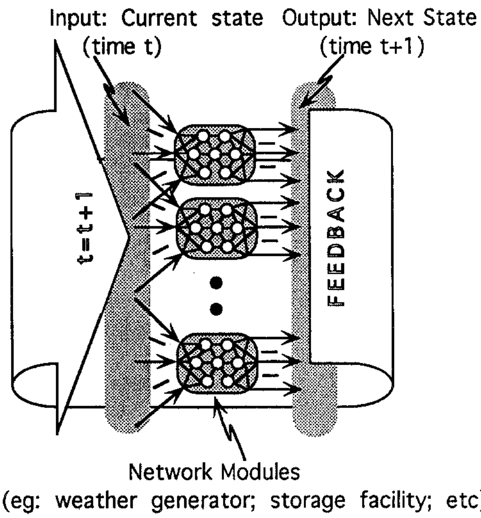
Input: Current state (time t)
Output: Next State (time t+1)
t=t+1
FEEDBACK
Network Modules (eg: weather generator; storage facility; etc)

FIG. 3. Modular Recursive Network for Simulation

subsequently, resulting in the output of the next state of the model. The feedback of information can be achieved most readily by using the same set of neurons for the inputs and outputs in the network. The time in the simulation is updated with each iteration. The model is allowed to run freely as such until the state of the model at the final point in simulation time is reached. During time, the behavior and performance of the model can be monitored by reading the appropriate outputs from the modules. Stochastic variance can be introduced to the model in the form of controlled random perturbations to the relevant inputs.

Since all components in the network operate in parallel (assuming the system is implemented on neural hardware), the rate at which a simulation proceeds (states per second) is independent of the complexity of a model. This speed can be several orders of magnitude faster than conventional approaches to simulation (Flood 1990b), with the advantage becoming more marked the more detailed the model. Existing experiments have been limited to software emulators running on a single-processor system, though future work will implement the system in neural hardware.

Feeding the output from the network back to the inputs in the cyclical manner described is often referred to as recursion. The problem with the recursive approach is that any small errors generated at the outputs quickly compound with each successive cycle, until the results have deviated completely from the correct answer. This can only be overcome if the neural network provides exact solutions at each cycle which, unfortunately, are not normally attainable with neural methods of processing. In the case of simulation, however, the interest is in modeling the characteristic behavior of the system, as opposed to its exact behavior. That is, it is only required to generate results that are representative of the systems behavior and performance. The validity of the neural-based simulator in this respect can be measured using the same techniques employed in conventional algorithmic approaches to simulation, such as by comparing plots of cumulative production predicted by the simulator with production measured on-site. The comparison can be made either visually or by using a statistical test such as the correlation coefficient. Pilot experiments indicate that the neural approach is capable of capturing the characteristic behavior and performance of equipment-intensive operations at least as accurately as a conventional algorithmic approach such as CYCLONE (Halpin and Woodhead 1976).

The simulation system must incorporate a comprehensive library of network modules representing the most fundamental types of process components required in model building. These modules need to be pretrained using a supervised scheme and should suffice for most problems. The system, however, also includes a facility that enables users to extend this library to match their particular needs. Developing a new module requires the user to provide training patterns representing the performance of the new process under a range of circumstances. This training data can be obtained from observations of ongoing processes, synthesized from historic data (such as published equipment and labor performance data), or consultation with a human expert [see the section on number, distribution, and format of training patterns in Flood and Kartam (1994)]. Whichever approach is adopted, each training pattern must comprise an input vector representing an observed set of values for those parameters in the system that may affect the process, and a corresponding output vector representing the state of the process at the next point in time. This data can be added to and/or modified with experience of the process under consideration, enabling additional training of the module. This approach to developing new modules has two advantages: (1) It enables the user to deal solely in familiar engineering related concepts, eliminating the need to become involved in any form of computer programming; and (2) the user does not need a rigorous understanding of the cause and effect relationships that determine the behavior of the process being modeled.

This simulation system makes predictions about the next state of the system based on its current state. Often, a more accurate prediction of the following state can be made using two or more previous states as inputs to the network. Such an approach has many potential applications in civil engineering, including: (1) Simulating dynamic loading on structures caused by, for example, winds; (2) extrapolating the time series of ground accelerations during a seismic event; (3) modeling the thermodynamic behavior of a building or of its components; (4) predicting the dynamic response of a structure to a sudden loading; and (5) projecting over a period of time the flow response of a drainage system to a storm.

Transitory Problems

Many problems in civil engineering undergo change over time so that what may be a valid solution at one time becomes inappropriate or invalid later. Many of the example problems considered fall into this category. For example: The production rate of an excavation system tends to increase with time as the operatives acquire more experience of the process; the dynamic response of a structure to a sudden load changes if the extent and distribution of fixed loading inside the building is altered; the flow response of a drain to a storm changes as new branches are added to the system, and the grading and surface cover of the catchment area are altered.

Sometimes it is possible to handle such variability in a problem by adding extra inputs to the neural network. For example, the effects of experience on the production rate of an excavation system can be taken into account by including an input to the network that registers the number of cycles of excavation work completed so far.

However, the way in which a problem changes and the consequences thereof cannot always be anticipated. Possible future changes to the surface cover of the catchment area of a storm drain is an example of this situation. Nevertheless, problems of this type can still be handled by neural networks provided the rate at which the problem changes is not too dramatic. Essentially, all that is required is to allow training of the network to continue after the network's implementation. As the circumstances of the problem change, new information is acquired and used to further the training of the network. This can proceed in several ways. Some or all patterns in the training set can be replaced as updated information becomes available or just the targeted outputs can be modified. The entire network can be subjected to this continual training or, in the case of an incremental training system such as the radial-Gaussian network (Flood 1991), selective hidden neurons can be retrained.

Optimization Problems

The design and construction of a facility, be it an office block or bridge, are tasks fraught with what are often termed optimization problems. These problems are characterized by the need to select a viable solution, from a large number of alternatives, that is optimal according to some measure of performance. Examples include: (1) Determining the optimum numbers and spacings for access shafts to a tunnel so that construction time is minimized; (2) arranging the cutting of material (such as reinforcement bars) so that wastage is minimized; and (3) scheduling activities so that the demand for equipment and labor resources are as constant as possible.

Typically, finding the overall optimum solution (rather than merely a viable or near optimal solution), is very difficult. Direct methods of finding an overall optimum solution are available for some simple problems: for example, calculus can be used for problems where the objective function is differentiable and does not include many locally optimal solutions; and, Johnson's method (Johnson 1954) can solve task sequencing problems that comprise no more than two (sometimes three) processes. For most practical problems, however, an indirect method of optimization must be used, which normally involves a search. Usually, indirect techniques are computationally expensive in the sense that the amount of time required to find the optimum (or near optimum) solution increases rapidly (often exponentially) with an increase in the number of independent variables in the problem. A compromise is often required, therefore, balancing computation time against the optimality of the solution found. For many problems, a solution sufficiently close to the optimum cannot be found in an acceptable period of time [see, for example, Issa et al. (1992)]. Neural networks can often overcome this limitation, providing quick solutions that are near optimal [see, for example, Hopfield and Tank (1985) and Flood (1990a)] or by providing first approximations that speed up the search process [see, for example, Issa et al. (1992)].

It is sometimes possible to develop a network to solve a class of optimization problems using a supervised training scheme. This was the approach adopted by Issa et al. (1992) in a system designed to assist in the selection of the optimum cable prestress tensions for guyed communications towers. In this case, the target outputs for a set of training examples were determined using a numerical solution algorithm (this algorithm was not an acceptable solution method on its own, since it required too many iterations to achieve an optimal answer). Once trained, the neural network was able to provide solutions that were good first approximations to the optimum answer. These could then be refined by application of a few iterations of the numerical procedure. The time required to solve a problem by finding a first approximation with the neural network and then refining it using the numerical approach was much less than using the numerical approach on its own.

Often, numerical and other search methods are too slow or do not provide consistently good results to be a practicable means of setting up training solutions. An alternative would be to consult an expert to provide good solutions for a range of situations, rather like collecting knowledge for an expert system. While often a valid approach, a drawback is that the performance of the system is limited to that of the expert's advice.

Another alternative would be to use a Hopfield network (Hopfield and Tank 1985), a type of neural network capable of solving a broad class of optimization problems. These devices comprise a collection of highly connected neurons that change their levels of activation over several iterations, moving toward a solution under the control of an energy function. An optimization problem can be implemented on a Hopfield network if its objective and constraints can be properly formulated into an energy function. Weights on the connections between the neurons are explicitly generated from this function, thus eliminating the time-consuming process of training. But the number of iterations required before the network settles on a solution can be numerous, particularly for large and complicated problems, thus a lot of computation time is needed. Additionally, a network usually becomes trapped in a local minimum, so the optimum solution is not found. Other drawbacks of this technique are that the number of neurons required in a network increases as a geometric function of the number of independent variables in the problem, and that establishing an energy function is not a straightforward task.

Another approach is to develop the weights in the network through a process of simulated evolution—an unsupervised training technique ideal for problems for which there are no good example solutions, but for which solutions produced by the network can be evaluated retrospectively. Consider the problem of determining an optimal sequence for a number of jobs that must go through a series of processes, with the following constraints: a process can operate on just one job at a time; each job must complete a process before moving onto the next; and the order for the processes is the

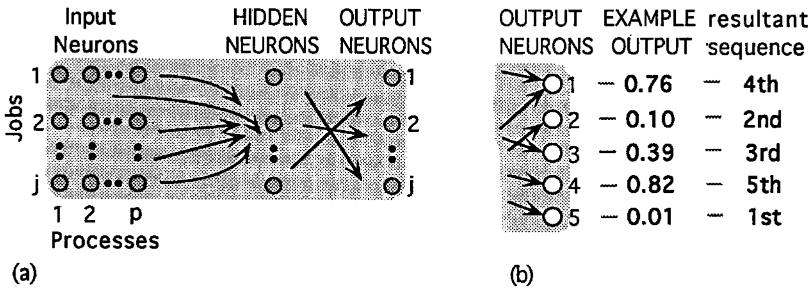

FIG. 4. Network for Task Sequencing: (*a*) Organization; (*b*) Interpretation of Output for Five-Job Problem

same for all jobs. The objective is to minimize the overall time required to complete the jobs. A network used to provide solutions to this problem can be structured as shown in Fig. 4(*a*). The input neurons are presented with the durations for each job at each process. At the output layer, there is one neuron for each job. When presented with a problem, the network responds by producing values across its outputs indicating an optimal sequence. The output with the lowest value denotes the first job to be processed; that with the next lowest value indicates the second job, and so on [see Fig. 4(*b*)].

A network is first set up with randomly selected values for the weights, along with a set of example problems without solutions. Each example problem is presented to the network in turn, and the resultant output is interpreted as a sequence for the jobs. The sequence indicated is evaluated in terms of the total time required for completion of the jobs. This is repeated for all the example problems and the average completion time is computed. The network is then adjusted by making small random adjustments to some of the weights. Following this, the performance of the network is reevaluated on the same set of example problems. If the modified network provides better solutions, then it is substituted for the original, otherwise the original network is kept. This process is repeated many times, gradually evolving towards a network that provides near optimal solutions. The network can then be tested on examples of the problem not used in the evolutionary process, to provide a more general evaluation of its performance. When the performance of the network is satisfactory, it can be used to suggest solutions to other similar sequencing problems. There are many enhancements that can be made to this approach, most notably, the maintaining of a number of networks and the crossbreeding of these networks to produce the networks in the new generation (Flood 1989).

Compared to Hopfield networks, simulated evolution has two advantages: first, the developer is not required to set up an energy function, and second, solutions can always be produced rapidly by the network. On the other hand, the time required to develop the network in the first place can be longer than that required to develop a Hopfield network.

The simulated evolution approach comes under the broader heading of genetic algorithms (Srinivas and Patnaik 1991), a numerical solution method that is gaining popularity as a means of developing neural networks due to its broad scope of application.

CONCLUSIONS

Compared to conventional digital computing techniques, and procedural and symbolic processing, neural networks are advantageous because they can learn from example and generalize solutions to new renderings of a problem, can adapt to fine changes in the nature of a problem, are tolerant to errors in the input data, can process information rapidly, and are readily transportable between computing systems. Certain conventional methods of computation can provide some of these advantages, but never more than a few. Neural networks do, however, suffer from a number of shortcomings, notably, a lack of precision, limited theory to assist in their design, no guarantee of success in finding an acceptable solution, and a limited ability to rationalize the solutions provided. Where appropriate, the advantages of neural networks can be combined with those of symbolic and procedural methods of computation in any of several hybrid schemes [see, for example, Kartam et al. (1994)].

Despite their limitations, neural networks offer a powerful means of solving poorly defined problems that have eluded solution by conventional digital computing techniques. Problems of this type are commonplace in civil engineering and already, within just a few years of the explosion in interest of this technology, successful solutions to seemingly intractable problems have been developed. The scope and pace of this accomplishment is likely to increase rapidly within civil engineering over the coming years.

The success of a neural-network implementation is dependent not just on the quality of the data used for training, but also on the type and structure of the neural network adopted, the method of training, and the way in which both input and output data are structured and interpreted. The effectiveness of the specific scheme chosen is, in turn, dependent on the type of problem to be solved. In the present paper, a number of different classes of problem were considered, and methods of solution were provided for each. There are, however, many variations on the types of problem discussed, and there are few hard and fast rules that should be adhered to when selecting a specific approach. Designing a successful approach for applying neural networks to a specific problem requires experience and imagination.

ACKNOWLEDGMENTS

The writers gratefully acknowledge the financial support to this research 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

Caudill, M. (1987). "Neural networks primer." *AI Expert*, 46–55.

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., the 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, UK.

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., and Kartam, N. (1994). "Neural networks in civil engineering: Systems and application." *J. Comp. in Civ. Engrg.*, ASCE, 8(2), 131–148.

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.

Halpin, D. W., and Woodhead, R. W. (1976). *Design of construction and process operations*. Wiley & Sons, New York, N.Y.

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

Hinton, G. E., and Sejnowski, T. J. (1986). "Learning and relearning in Boltzmann machines." *Parallel Distributed Processing*, Vol. 1, D. E. Rumelhart and J. McClelland, eds., MIT Press, Cambridge, Mass., 282–317.

Hopfield, J. (1982). "Neural networks and physical systems with emergent collective computational properties." *Proc., Nat. Acad. of Sci.*, Vol. 79, 2554–2558.

Hopfield, J., and Tank, D. (1985). "Neural computation of decisions optimization problems." *Biological Cybernetics*, Vol. 53, 141–152.

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.

Johnson, S. M. (1954). "Optimal two- and three-stage production schedules with set-up times included." *Nav. Res. Logist. Q.*, 1, 61–68.

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.

Karshenas, S., and Feng, X. (1992). "Application of neural networks in earthmoving equipment production estimating." *Proc., 8th Conf. in Comp. in Civ. 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.

Pao, Y. (1989). *Adaptive pattern recognition and neural networks*. Addison-Wesley Publishing Co., Reading, Mass.

Pilcher, R., and Flood, I. (1984). "The use of simulation models in construction." *Proc., Inst. of Civ. Engrs., Vol. 76, Part 1, Des. and Constr.*, ICE, London, England, 1241–1247.

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.

Srinivas, M., and Patnaik, L. (1991). "Learning neural network weights using genetic algorithms—improved performance by search-space reduction." *Proc., Int. Joint Conf. on Neural Networks, IEEE and INNS*, Neuro-Dynamics II, Vol. 3, Singapore, 2331–2336.

Udpa, L., and Udpa, S. S. (1991). "Neural networks for the classification of nondestructive evaluation signals." *IEE Proc., -F*, 138(1), 41–45.

Wong, F., Tung, A., and Dong, W. (1992). "Seismic hazard prediction using neural nets." *Proc., 10th World Conf. on Earthquake Engrg.*, 339–343.