Review

Neural Networks in Mechanical System Simulation, Identification, and Assessment

Thomas L. Paez
Experimental Structural Dynamics Department
Sandia National Laboratories
Albuquerque, NM 87185

Artificial neural networks (ANNs) have been used in the solution of a variety of mechanical system design, analysis, and control problems. This paper describes the ANNs that have been most frequently used in mechanical system applications. It also summarizes some of the applications that have been developed for ANNs, and briefly reviews the literature where descriptions of the developments and applications can be found. Some recommendations regarding ANN applications in mechanical system simulation, identification, and assessment are provided. © *1993 John Wiley & Sons, Inc.*

INTRODUCTION

Artificial neural networks (ANNs) were first introduced in the 1940s. Their original purpose was to simulate aspects of brain behavior and function using simple, yet realistic, elements. Some reviews of the early studies and their intent are given in many texts and articles, for example, Freeman and Skapura (1991) and Vanluchene and Sun (1990).

Much of the terminology associated with ANNs was developed in the original studies and is still used. For example, terms such as neuron, neural network, and activation, to be described in the following, originated in early ANN applications and describe elements and functions of the brain.

Though it was originally used to simulate brain function and behavior, engineering and science applications have simply taken advantage of the ANNs' ability to simulate highly complex mathematical mappings. The ANN provides a different method for approaching problems in engineering analysis than those traditionally used. A clear motivation is given by Jones, Lee, Qian et al. (1990):

A young child can learn to operate a complex nonlinear process, walking or riding a bicycle for instance, with no knowledge of physics or differential equations. A juggler can balance an inverted broom stick with only a coarse knowledge of the state of the system. A human being seems to be able to act with reasonable precision in a complex highly interactive world with partial or partially incorrect information. One wonders, then, if the traditional deductive approach of physicists and engineers to prediction and control cannot be augmented by a more inductive point of view. Does one, for example, need to know the details of plasma instability in order to predict and control it? Can the behavior of a chaotic process be predicted with no initial knowledge of the dynamics or dimensionality of the system? Taking life as an existence of proof, we are motivated to look at some inductive and adaptive systems and their interaction with complex nonlinear phenomena.

Shock and Vibration, Vol. 1, No. 2, pp. 177–199 (1993)
© 1993 John Wiley & Sons, Inc.

CCC 1070-9622/93/020177-23

The techniques most frequently used today for engineering problem analysis involve the description of the physical principles governing the phenomenon of interest, the use of these principles to develop the differential equations governing the phenomenon, and analysis of the differential equations to establish the solution. Such techniques are phenomenological. There are alternatives to the traditional analytical approaches. Some alternatives that are nonphenomenological are canonical variate analysis (Larimore, 1983) and ANNs. The objectives of this article are to:

1. provide a general description of ANNs;
2. describe in more detail, the operation of four ANNs;
3. give some examples of their use by reviewing some of the literature that describes developments and applications of ANNs;
4. discuss the simulation, identification, and assessment of mechanical systems with ANNs; and
5. make some recommendations for applications of ANNs.

Even though ANNs have only been used in mechanical system applications for about 5 years, there are some articles that include literature reviews that discuss applications to date. Among these is one by Hajela and Berke (1992) where several types of ANNs and applications that are not covered in this article are discussed. They discuss counter propagation networks, Hopfield networks, adaptive resonance theory networks, and mechanical system applications that use them. Another article that includes a literature review is the one by Vanluchene and Sun (1990). They discuss several articles that cover information of interest in mechanical applications.

ANNs

An ANN is an assembly of a (potentially large) number of interconnected processing units called neurons, processing elements, or nodes. The structure, rules of operation, and environment for the ANN are discussed in detail in Chapter 2 of Rumelhart, Hinton, and McClelland (1986). ANNs are intended to perform functions that involve the processing of input data to yield output data, or, in other terms, functions that map input data, $\{x\}$, to output data, $\{z\} = g(\{x\})$, where $g(.)$ is a deterministic function. An ANN performs its function in any of a number of specific frameworks (some of which are described in the following) by developing an approximation for $\{z\} = g(\{x\})$ that can be expressed $\{y\} = h(\{x\})$, where $\{y\}$ is the ANN approximation to $\{z\}$. In the mechanical system framework, the input data, $\{x\}$, and the output data, $\{z\}$, can consist of measures of mechanical system excitations, measures of responses, system parameters, or any combination of the three. The purpose of an ANN can be the simulation of system behavior, the determination of some measure of system excitation, the determination of system parameters, the characterization of some aspect of system behavior, or the classification of the input presented to the ANN. The overall ANN is characterized by its size, geometry, pattern of connectivity, the rules of operation of its neurons, and the direction of flow of input stimuli presented to the ANN and the rules for propagation of the stimuli.

A neuron in an ANN is typically a simple element that operates on externally supplied inputs or inputs supplied from other neurons. The rules of operation of a neuron establish how it processes the inputs presented to it, how it calculates an activation for the neuron based on the inputs, and how it calculates an output based on the activation.

Use of the neural network requires that we be able to train it to perform a function; therefore, it is required that we establish a learning rule that can be used to teach the neural network to perform the operation intended. Finally, an ANN is usually intended to operate in a preestablished environment. This means that there is a family of inputs to an ANN that can be operated on to produce (or mapped to) outputs that accurately reflect the operations intended in the ANN.

An ANN usually consists of at least an input layer of neurons, and an output layer of neurons, and many ANNs include one or more hidden layers of neurons. Figure 1 is a schematic of a generic ANN including one hidden layer of neurons. ANNs are used to perform various tasks, and the framework in which they may perform these tasks can be described as follows (Fig. 1). This description outlines what is known as feedforward operation of an ANN.

A vector of inputs (or stimuli or signals), $\{x(t)\}$, is presented to the ANN. [The index $t(t = 1, 2, \ldots)$ is included for situations when it is important to consider the temporal order of pre-

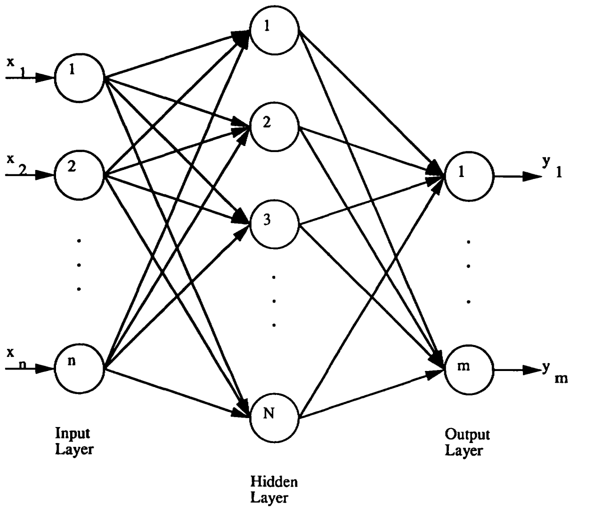

FIGURE 1 Framework for a neural network with *n* input nodes, *N* hidden level neurons, and *m* output neurons.

sentation of the inputs.] Each element in the vector enters the ANN through an input node. The layer of nodes that accepts the inputs is called the input layer. The input nodes may operate on the input vector elements, for example, by normalizing the inputs, or they may simply distribute the inputs to the following layer. The following layer is the first layer of hidden nodes if the ANN has been defined to include such a layer. Otherwise, it is the output layer. Typically, the outputs of the input layer are distributed to all the neurons in the following layer. (Exceptions occur in some implementations, and with some types of ANNs. In these cases, the outputs from the input layer may be distributed to only a fraction of the neurons in the immediately succeeding layer, or some outputs of the input layer may be distributed to a layer beyond the immediately succeeding layer.) The inputs to the layer that succeeds the input layer are processed in the individual neurons of the layer, and each neuron produces in own output. The output of a neuron in a layer may be influenced only by activities of parameters involving the neuron itself, or it may be influenced by the activities or parameters of all the neurons in the layer; the former is the usual case. This process of inputting stimuli to a layer and the generation of outputs from the layer continues through the hidden layers (if there are any) until the output layer of neurons is reached. The stimuli presented to the output layer are processed as described above for previous layers, and the output layer produces the ANN output vector, $\{y(t)\}$.

The processing that takes place within each neuron in an ANN (and, perhaps, throughout a layer of neurons, simultaneously) depends on the type and function of the ANN. The inputs to a neuron are related to its single output through algebraic equations, differential equations, or both. However, the operation of an ANN most frequently involves application of the following specific and generic algebraic steps. These steps are shown schematically in Fig. 2. All the outputs from a preceding layer of neurons (a hidden layer or the input layer) in an ANN are presented to a neuron as input. Each presented input is multiplied times a (potentially adjustable) weight value, and the resulting products are summed. This yields a quantity known as the net input to the neuron. If we let o_i, $i = 1, \ldots, n$, denote the ith output from the previous layer of an ANN, w_i, $i = 1, \ldots, n$, denote the ith weighting value in a particular neuron, and net denote the net input to a neuron, then the net input is computed

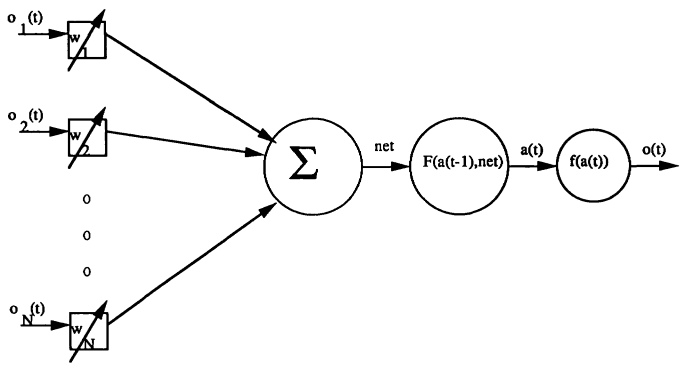

FIGURE 2 Model for a neuron in an artificial neural network (ANN).

with the equation

$$\text{net} = \sum_{i=1}^{n} w_i o_i. \tag{1}$$

This is simply the dot product between the vectors $\{w\}$ and $\{o\}$ of elements, w_i, o_i, $i = 1, \ldots, n$, respectively. Next, the neuron performs a functional operation on the net input and the previous value of the neuron activation to establish the present value of the neuron activation. This operation can be expressed

$$a(t) = F[a(t-1), \text{net}(t)] \tag{2}$$

where the concept of time and sequence of operations has been explicitly accounted for through the use of the index, t. The activation characterizes the "level of activity" of the neuron. Sometimes, the function that establishes the current activation of a neuron is a threshold function or a function of another nontrivial form, but more frequently, it is simply an identity that equates the current value of activation to the net input. (Because of this fact, many descriptions of neuron operation ignore the activation, and simply pass the net input to the following step.) It will be assumed in all the following discussions of ANNs that the activation function is an identify that simply equates the activation to the net value. Finally, the neuron performs a functional operation on the activation to produce the neuron output, o. This operation can be expressed

$$o = f[a(t)] = f(\text{net}). \tag{3}$$

The function $f(.)$ is often called the activation function. As mentioned in the previous paragraph, this output is distributed to the neurons in the succeeding layer in the ANN, or, if the neuron under consideration is in the output layer of the ANN, it is interpreted as an output of the ANN.

Note that the operations described above can be written in matrix and vector function form. Consider an ANN with n inputs, M hidden layers with N neurons in each layer, and m outputs (refer to Fig. 3.) Assume that the input layer simply distributes the inputs to the following layers. (If some normalization is required, it can be done before the inputs are presented to the ANN.) Let the vector $\{x(t)\}$ denote the inputs to the ANN at time index t, the vector $\{y(t)\}$ denote the outputs

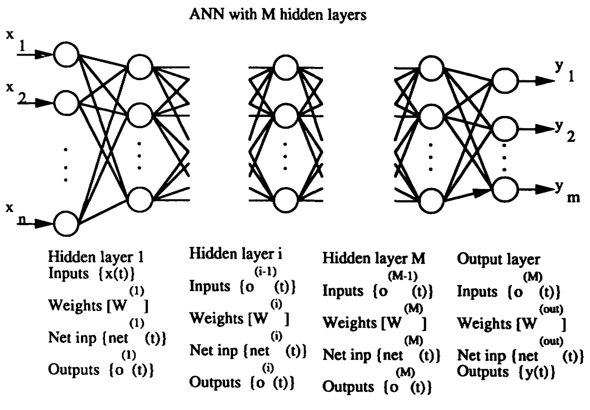

FIGURE 3 Schematic for description of matrix operations in an ANN.

of the ANN, the matrix $[W^{(i)}]$, $i = 1, \ldots, M$, denote the weights of the ith hidden layer, the matrix $[W^{\text{out}}]$ denote the weights of the output layer, the vector $\{\text{net}^{(i)}(t)\}$, $i = 1, \ldots, M$, denote the net inputs to the neurons in the ith hidden layer, the vector $\{\text{net}^{(\text{out})}(t)\}$ denote the net inputs to the neurons in the output layer, and the vector $\{o^{(i)}(t)\}$ denote the outputs of the neurons on the ith hidden layer of the ANN. The jth row, kth column element of a weight matrix for a particular layer is the weight on the kth output coming from the previous layer to the jth element in the current layer. We can establish the net inputs to the first hidden layer with the formula

$$\{\text{net}^{(1)}(t)\} = [W^{(1)}]\{x(t)\}, \quad t = 1, 2, \ldots \tag{4}$$

Based on the ANN size, the matrix $[W^{(1)}]$ must have the dimensions $N \times n$. The output of the first hidden layer is

$$\{o^{(1)}(t)\} = f_1[\{\text{net}^{(1)}(t)\}], \quad t = 1, 2, \ldots \tag{5}$$

where $f_1(.)$ is the vector activation function for the first hidden layer of neurons. The net inputs to the ith hidden layer are

$$\{\text{net}^{(i)}(t)\} = [W^{(i)}]\{o^{(i-1)}(t)\}, \quad t = 1, 2, \ldots \tag{6}$$

Based on the ANN size, the matrix $[W^{(i)}]$ must have the dimensions $N \times N$. The output of the ith hidden layer is

$$\{o^{(i)}(t)\} = f_i[\{\text{net}^{(i)}(t)\}], \quad t = 1, 2, \ldots \tag{7}$$

where $f_i(.)$ is the vector activation function for the ith hidden layer of neurons. Finally, the net inputs to the output layer are computed from

$$\{\text{net}^{(\text{out})}(t)\} = [W^{(\text{out})}]\{o^{(M)}(t)\}, \quad t = 1, 2, \ldots \tag{8}$$

Based on the ANN size, the matrix $[W^{(\text{out})}]$ must have the dimensions $m \times N$. The output of the ANN is computed from

$$\{y(t)\} = f_{\text{out}}[\{\text{net}^{(\text{out})}(t)\}], \quad t = 1, 2, \ldots \tag{9}$$

where $f_{\text{out}}(.)$ is the vector activation function for the output layer of neurons. The subscript on each activation function, $f_j(.)$, implies that the functional form and parameters can differ from layer to layer. In spite of this, the functional form and parameters are frequently taken to be the same throughout the ANN.

As mentioned previously, the above operations describe feedforward operation of an ANN. Clearly, the feedforward operation of an ANN maps inputs $\{x(t)\}$ to outputs $\{y(t)\}$. Many of the functions that an ANN might perform with feedforward operations (particularly those of interest in mechanical system applications) are discussed in later sections. In order to accurately perform an operation, an ANN must be trained using $\{x\}$ and $\{z\}$ data pairs. A collection of data used to train an ANN can be denoted $\{x_j\}$, $\{z_j\}$, $j = 1, \ldots R$. Each matched pair $\{x_j\}$, $\{z_j\}$ is known as an exemplar because it is an example of correct behavior. The learning rule used to train an ANN differs according to the type of ANN under consideration. Four types of ANNs and their learning rules are described in the following sections. In most applications, training of an ANN involves modification of the weights in the weight matrices that cause the ANN to accurately perform its intended function. Typically, the data used to train an ANN come from laboratory tests, field tests, or other (more complicated, more time consuming) analyses.

There are two general modes in which ANNs learn their functions. These are supervised and unsupervised learning. In supervised learning, an ANN is presented with input–output pairs and trained to emulate their behavior; this is the type of learning that will be emphasized in this article. In unsupervised learning, the ANN is first trained to represent the input training vectors, then the representation of a class of outputs may be developed. Some ANNs that accommodate unsupervised learning are counterpropagation networks and self-organizing maps. In general, the activities involved in training an ANN to represent the characteristics of a class of inputs are referred to as self-organization. Several chapters are devoted to these subjects in Freeman and Skapura (1991).

The data, $\{x_j\}$, $\{z_j\}$, $j = 1, \ldots, R$, used to train an ANN come from an environment with some specification. In particular, when the elements of the vectors $\{x_j\}$ and $\{z_j\}$ are continuous valued, the vectors $\{x_j\}$, $j = 1, \ldots, R$, span a space with some dimension, have some density over the space, etc., and the same is true of the vectors $\{z_j\}$, $j = 1, \ldots, R$. When training is performed on an ANN, it learns to map input vectors $\{x\}$ into output vectors $\{z\}$, and it is usually hoped that the ANN learns to correctly interpolate among the vectors $\{z_j\}$, $j = 1, \ldots R$, when presented with a vector from among the $\{x_j\}$, $j = 1, \ldots, R$. Hornik, Stinchcombe, and White (1989), show that if an ANN with at least one hidden layer contains sufficient neurons, is sufficiently trained, and the underlying mapping is deterministic, then it will perform interpolations correctly, and it can be made arbitrarily accurate. It is said that the $\{x_j\}$ and $\{z_j\}$, $j = 1, \ldots, R$, data define the environment of the ANN, and the ANN will perform satisfactorily for $\{x\}$, $\{z\}$ pairs within the environment. It is usually assumed that the extrapolation capability of an ANN is questionable, as indeed it should be. There is usually no reason to believe that an ANN should perform well in an environment where it has not been trained.

There are many practical issues involving the use of ANNs, and an important one among these is size of the ANN. For most frequently used ANNs, relations are not available to quantify the increase in accuracy attained by increasing the number of neurons in an ANN for a problem whose input environment spans a space with fixed dimension. In practical situations, investigators often start with an ANN and increase its size until the increase in accuracy of the input–output mapping diminishes to an acceptable level.

Though the preceding paragraphs describe the general operation of an ANN, a neuron within the ANN, and the environment in which an ANN operates, it must be emphasized that there are many types of ANNs, and the specific structure, rules for operation of the overall network, rules for operation of the neurons in the network, and environment in which the network will operate, may all differ in their details. For more detailed descriptions of the ANNs described here and other ANNs, refer to Rumelhart, Hinton, and McClelland (1986) and Freeman and Skapura (1991).

The following paragraphs describe the operation and use of four specific types of ANNs. These are the backpropagation network, the probabilistic neural network, the associative memory, and the radial basis function network. There are many other types of networks, including madeline networks, counterpropagation networks, self-organizing maps, and networks based on adaptive resonance theory. However, these will not be described in this article.

Backpropagation Network

The ANN that appears to be used most frequently in all applications is the backpropagation network (BPN), sometimes called the feedforward/backpropagation network. The BPN is an example of a network like the one shown in Fig. 3, with neurons like the ones shown in Fig. 2, and described in the previous section. The BPN has an input layer, an output layer, and typically one or more hidden layers. It derives its name from the fact that when training is performed on the weights in the network, it proceeds from the output layer to the final hidden layer, to the previous one, etc., until it reaches the first hidden layer.

The number of neurons in the input layer is the number required to accept the information appropriate to the problem. (The problem framework usually dictates the appropriate inputs, but sometimes, particularly in experimental situations, it is difficult to establish precisely what are the appropriate physical inputs. In some applications, some fraction of the known physical parameters or parametric measures of the physical inputs are used as the ANN inputs.) The number of neurons in the output layer of the BPN is the number required to provide the information sought from the BPN.

The number of hidden layers and the number of neurons in each hidden layer of a BPN is arbitrary. However, it is intuitively clear that as the complexity of a mapping from BPN inputs to outputs increases, the number of nodes in the hidden layers must increase in order for the mapping to be modeled accurately. The number of weights in the hidden layers is a measure of the size of a BPN. There is no universal rule for choosing the number of hidden layers and neurons in each

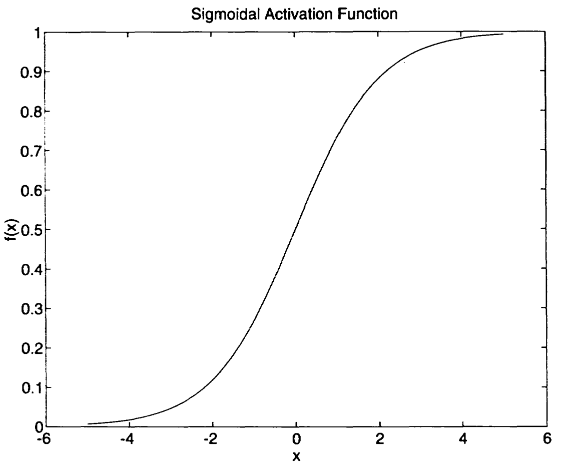

FIGURE 4 Sigmoidal activation function used to process the net input to a neuron in a BPN.

layer in a BPN for a specific problem representation. However, some investigators have found that one means for sizing a BPN is to start at one size and increase it until the error in the representation diminishes to a satisfactory level after training. Many complicated mappings can be accurately simulated with a BPN that has only one hidden layer.

The activation functions relating net input to neuron output in an ANN have been defined in different ways in various networks. The most commonly used activation functions are step functions, linear functions, and sigmoidal functions. The most frequently used activation function is the sigmoidal curve whose specific form is

$$f(x) = \frac{1}{1 + e^{-x}}, \quad -\infty < x < \infty. \tag{10}$$

This function is graphed in Fig. 4. Note that the curve has an ordinate value between zero and one, and that the ordinate value is only sensitive to abscissa values in a limited range, say $(-3, 3)$. The linear activation curve appears limited in its ability to represent complex, and perhaps nonlinear, input to output mappings, and is probably not as good as the sigmoidal curve in mechanical system applications. Use of the step function activation curve clearly makes it impossible to generate a continuous valued output mapping.

It appears that a critical element in defining a BPN with the capability to accurately model a realistic mapping is normalization of the magnitudes of inputs and outputs. During training (to be described in the following paragraphs), the stimuli to individual input neurons, and the responses to be used in training BPN outputs, can be normalized to preestablished ranges. For example, stimuli to input nodes might be scaled into the range $(-3, 3)$, and training responses from the output neurons might be scaled to the range $(0.1, 0.9)$. This clearly requires some foreknowledge of the environment defined by the $\{x_j\}, \{z_j\}, j = 1, \ldots, R$. When a BPN is trained with normalized inputs and outputs, later applications with actual data in the feedforward mode require that the inputs be normalized and the BPN outputs be denormalized upon output from the BPN. Normalization of the inputs and use of the activation function in Eq. 10 is equivalent to use of the actual inputs with a more general sigmoidal curve.

The objective of training the BPN is to make its operation mimic the mapping behavior $\{z\} = g(\{x\})$ using the training set, $\{x_j\}, \{z_j\}, j = 1, \ldots, R$. This is usually accomplished with an approach known as the generalized delta rule. (See, for example, Chapter 8 of Rumelhart, Hinton, and McClelland, 1986.) The generalized delta rule (or any rule used to train a BPN) seeks to select the weights in the BPN that minimize some measure of the difference between $g(\{x\})$, the mapping that is to be modeled, and $h(\{x\})$, the mapping provided by the BPN. The steps in the generalized delta rule are as follows.

1. Select dimensions, m, n, M, N, for the BPN, randomize the elements, $w_{jk}^{(i)}$, of all the weight matrices, and select a learning rate, η. [This is a number, usually chosen in the range (0.05, 0.20), that establishes the fraction of the correction to be applied to a weight during a training cycle.]
2. Present a training vector $\{x_j\}, j = 1, \ldots, R$, to the BPN.
3. Propagate it through the BPN using Eqs. (4)–(9) to compute the BPN output, $\{y_j\}$.
4. Compare the computed BPN output, $\{y_j\}$, to the training output, $\{z_j\}$, by computing the error and the squared error. The error, $\{\varepsilon_j\}$, is the vector of the differences between the elements of $\{z_j\}$ and $\{y_j\}$. The scalar squared error, ε_j^2, is one-half the sum of the squares of the differences between the elements of $\{z_j\}$ and $\{y_j\}$.
5. Establish the rate of change of the squared error with respect to each of the weights in the BPN, $\partial\varepsilon_j^2/\partial w_{jk}^{(i)}$, using Eqs. (4)–(9). (This is minus one times the dot product between the error vector, $\{\varepsilon_j\}$, and the partial derivative of the BPN output with respect to $w_{jk}^{(i)}$, $\partial\{y_j\}/\partial w_{jk}^{(i)}$.)
6. Multiply the rate of change of the squared error with respect to each of the weights in the BPN by the learning rate, η, to create the weight corrections $\delta w_{jk}^{(i)}$.
7. Subtract the weight corrections from the weights, $w_{jk}^{(i)}$, to update the weights in the BPN.
8. Repeat steps 2 through 7, until the squared error reaches a satisfactory value.

The training steps described here involve the least mean square (LMS) method of error minimization. This method is discussed in detail in Widrow and Stearns (1985). The identification of the optimal BPN weights can be accomplished by other error minimization techniques including, for example, gradient search techniques, simulated annealing (Press, Flannery, Teukolsky, and Vetterling, 1988), and the genetic algorithm (Holland, 1992). The latter two are global techniques, and can be very useful when the error surface associated with the BPN weights is not unimodal, that is, has multiple extrema.

The operations involved in the identification of the weights in a BPN (or other ANN) may seem quite similar to the operations of a nonlinear regression, and, in fact, they often are. However, the use of large BPNs and the use of BPNs in real time applications introduce special problems that may not be encountered in nonlinear regression. Werbos (1989) discusses this issue and other issues related to identification and control with ANNs.

As stated previously, the great majority of applications of ANNs to mechanical system problem solutions use BPNs. The applications of BPNs to mechanical systems range from static and dynamic system simulation, to structural health monitoring, to classification of the characteristics of system response. A sequence of papers by Hajela and Berke (1991, 1992) and Berke and Hajela (1992) considers the simulation of static mechanical system response to applied loading using BPNs and other types of ANNs (counterpropagation networks, Hopfield networks, and adaptive resonance theory networks). Their purpose in establishing the simulation is to use the outputs of the BPNs in mechanical system optimization. When an ANN can be used to accurately model the behavior of a mechanical system, the determination of an optimum design will be much more efficient because the ANN yields its output much faster than another analysis, for example, a finite element analysis. Their examples concentrate on truss systems. They point out that once neural networks are developed to establish optimum designs in structural systems, the optimal designs can be used to train other ANNs to provide specific information about structure geometries. For example, when a system with preestablished general geometry is to be designed and some of the dimensions are known, then an ANN can be used to specify the remaining dimensions of an optimum structure.

An area where the BPN can clearly be used efficiently is the simulation of the behavior of mechanical systems. Before an ANN can be used to simulate system behavior, it must be trained using the techniques discussed. Among the applications where identification and simulation of system behavior are required are structural control and identification of system parameters. Some articles that deal with the identification of system models for use in the control framework are the ones by Goh and Noakes (1993) and Wang and Miu (1991). The former article uses a BPN in a recurrent framework (see recurrent ANNs discussion) to simulate the response behavior of the system to be controlled, in a nonlinear output feedback controller. It is assumed that the general structure of the system to be controlled is unknown, and its internal states are not observed directly. Examples are shown where very good accuracy in the system simulation is achieved. The latter article uses a BPN to control the motion of a system whose motion is simulated with a BPN. They show that the motion of the inverted pendulum can be controlled using the scheme they developed.

The identified parameters of a BPN can also be used to infer the parameters of a structure. Riva and Giorcelli (1992) point out that a one-to-one comparison can be made between the expanded form of an autoregressive-moving average (ARMA) or nonlinear ARMA with exogenous terms (NARMAX) model, and the model of a BPN. Once the weights in the BPN are identified they can be used to compute the linear or nonlinear parameters of a mechanical system. For example, the modal parameters of a system can be inferred.

The BPN is useful for the prediction of system response character at a time Δt in the future (where Δt is small). The BPN input is the forcing function at time t and the state of system motion at time t, and the output of the BPN is the state of system motion at time $t + \Delta t$. (See recurrent ANNs section.) Yamamoto (1992) obtained a very accurate model for predicting the response of a single degree-of-freedom system using this approach.

Wu, Ghaboussi, and Garrett (1992) attempted to use a BPN to identify damage in a three degree-of-freedom system. The inputs to the BPN were the Fourier transforms of the responses at various degrees of freedom of the structure excited by some earthquake accelerations. The outputs of the BPN were the postevent stiffnesses in the structural members. They achieved partial success in their attempts to identify damage, in the sense that they could frequently, accurately predict damage in the location where it occurred. However, they also found that sometimes when damage occurred, it was difficult to determine its location.

The objective of the article by Xirouchakis and Norris (1991) was the classification of column buckling modes. They point out that the postprocessing of finite element analysis results often require the interpretation of complicated numerical data. They suggest that an ANN can be useful in the interpretation process. They trained several BPNs to perform a sequence of operations related to classification. When combined in a specific manner, these can be used to classify column buckling modes. Further, they were able to estimate the effective column length ratio of a buckled column.

In addition to presenting a brief discussion of several types of ANNs, the article by Vanluchene and Sun (1990) presents three examples of the use of BPNs in mechanical system analysis. The first problem they solved involves the identification of the location of application of a static load to a beam. Second, they show that a BPN can be used to specify the design dimensions of a reinforced concrete beam when load and geometry information are provided. Third, they trained a BPN to identify the maximum moments and their locations in a simply supported plate subjected to concentrated load. The location of the load is the input to the BPN.

It is clear that there are numerous applications for the BPN in mechanical system analysis. A matrix of the general classes of applications is presented and discussed in a later section. An important strength of the BPN is that it is a universal approximator, as discussed in Hornik, Stinchcombe, and White (1989). This indicates that if the mechanical system under consideration is modeled with a BPN that has adequate size, adequate training, and is deterministic, then a BPN should be able to accurately simulate its behavior. The primary weakness of a BPN is that it may require a substantial amount of training, and the training cycles execute relatively slowly. (This is clearly a function of the size of a BPN.) This may limit the usefulness of the BPN in real-time applications where the system under consideration varies rapidly, or it may introduce the need for special training procedures.

Probabilistic Neural Network

The probabilistic neural network (PNN) is an ANN that is capable of performing pattern classification on data sequences presented to it. The PNN was first proposed in an article by Specht (1990). The PNN makes decisions in the following way. Given a vector of input information, the PNN decides whether the data in the vector belong to class A or class B. This can be stated more formally. Let θ denote the state of nature. We seek to make a decision, $d(\{x\})$, regarding the state of nature, either $\theta = \theta_A$ or $\theta = \theta_B$, based on a measured vector, $\{x\}$.

A framework for making such a decision is the Bayes decision rule. It states that

$$d(\{x\}) = \begin{cases} \theta_A, & \text{if} \quad h_A L_A f_A(\{x\}) > h_B L_B f_B(\{x\}) \\ \theta_B, & \text{if} \quad h_A L_A f_A(\{x\}) < h_B L_B f_B(\{x\}) \end{cases}. \tag{11}$$

$f_A(\{x\})$ and $f_B(\{x\})$ are the joint probability density functions (pdf) of the random vector $\{X\}$ (a vector of random variables X_i, $i = 1, \ldots, n$) given that the state of nature is $\theta = \theta_A$ or $\theta = \theta_B$, respectively. L_A and L_B are the loss functions associated with the decisions $d(\{x\}) = \theta_B$, when in fact $\theta = \theta_A$, and $d(\{x\}) = \theta_A$, when in fact $\theta = \theta_B$, respectively. h_A and $h_B = 1 - h_A$ are the a priori probabilities of occurrence of patterns from categories A and B, respectively. The boundary between the regions in which $d(\{x\}) = \theta_A$ and $d(\{x\}) = \theta_B$ is given by the $\{x\}$ that satisfy the equation

$$h_A L_A f_A(\{x\}) = h_B L_B f_B(\{x\}). \tag{12}$$

When the elements in this equation can be specified, then the decision defined in Eq. (11) can be made. The greatest difficulty is in establishing estimates for the joint pdfs.

A means for developing estimates for the joint pdfs is given in Specht (1990), based on articles by Parzen (1962) and Cacoullos (1966). The estimate is given as follows. Let $\{x_{Ai}\}$, $i = 1, \ldots, R_A$, be a sequence of vector realizations of the random vector $\{X\}$, when $\theta = \theta_A$. Then a joint pdf estimate for the random vector $\{X\}$ is given by

$$f_A(\{x\}) = \frac{1}{(2\pi)^{n/2}\sigma^n} \frac{1}{R_A} \sum_{i=1}^{R_A} \exp \left[-\frac{1}{2\sigma^2} \|\{x\} - \{x_{Ai}\}\|^2 \right] \tag{13}$$

where n is the dimension of the random vector $\{X\}$, and σ is a smoothing parameter. This is simply a normalized sum of multivariate normal pdfs, each with variance parameter, σ^2, drawn about one of the realizations, $\{x_{Ai}\}$. When σ is small and R_A is small, and the vectors $\{x_{Ai}\}$, $i = 1, \ldots, R_A$, are not densely packed in space, then $f_A(\{x\})$ appears like a collection of spikes in space. As σ and R_A increase, and as the spatial density of the $\{x_{Ai}\}$, $i = 1, \ldots, R_A$, increases, the joint pdf estimate appears smoother. Examples of joint pdf estimates of bivariate vectors are shown in Figs. 5(a) and 5(b). In both cases, there are five vector realizations represented in the pdf estimate. Figure 5(a) shows a representation where $\sigma = 0.4$ is small relative to the sample density; Fig. 5(b) shows a representation where $\sigma = 0.8$ is large relative to the sample density.

In order to make decisions like the one described above in an ANN framework, we construct a PNN as shown in Fig. 6. The PNN is used as follows. First, before presenting any in-

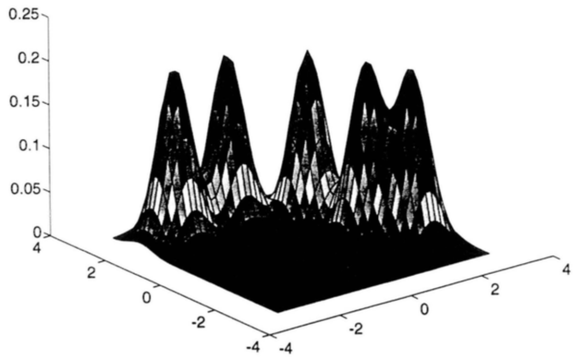

FIGURE 5(a) Estimate of the pdf based on five bivariate vector realizations ($\sigma = 0.4$).

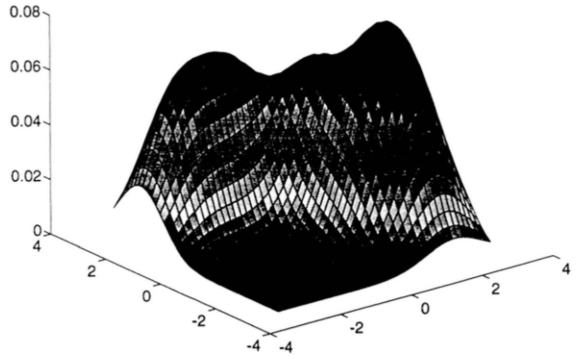

FIGURE 5(b) Estimate of the pdf based on five bivariate vector realizations ($\sigma = 0.8$).

put vector, $\{x\}$, to the PNN, we normalize its length to one, that is, normalize $\{x\}$ so that $\|x\| = 1$. Let $\{x_{Ai}\}$, $i = 1, \ldots, R_A$, be a sequence of vectors from the training set in which $\theta = \theta_A$, and let $\{x_{Bi}\}$, $i = 1, \ldots, R_B$, be a sequence of vectors from the training set in which $\theta = \theta_B$. Define the first hidden layer of the PNN as one that contains $N = R_A + R_B$ neurons. (This layer is called the pattern layer in PNN terminology.) Define the weights in the first R_A hidden layer neurons such that the elements in the row vector of weights in neuron j equal the normalized values of the training vector $\{x_{Aj}\}$, that is,

$$w_{jk} = \frac{x_{Akj}}{\|x_{Aj}\|}, \quad j = 1, \ldots, R_A, \quad k = 1, \ldots, n \tag{14}$$

where x_{Akj} is the kth element in the training vector $\{x_{Aj}\}$. Similarly, define the weights in the hidden layer neurons indexed $R_A + 1$ through $R_A + R_B$ such that the elements in the row vector of weights in neuron j equal the normalized values of the training vector $\{x_{Bj}\}$. These operations constitute the training of the PNN. This sort of training is obviously much faster than any iterative approach to ANN training.

Define the net inputs to the neurons in the first hidden layer in the usual way, according to Eq. (4). Because the input vector is normalized, and the weights are normalized, the net input to each neuron is a quantity in the interval [0, 1]. (If the input vector is orthogonal to the weight vector, then net = 0; if the input vector equals the weight vector, then net = 1.) The output of each neuron in the first hidden layer is the result of an operation on the net input to the neuron defined by

$$o^{(1)} = \exp\left[\frac{1}{\sigma^2}(\text{net} - 1)\right]. \tag{15}$$

Note that this is identical to the multivariate normal pdf (without the normalizing coefficient)

$$o^{(1)} = \exp\left[-\frac{1}{2\sigma^2}\|\{w\} - \{x\}\|^2\right] \tag{16}$$

where $\{w\}$ is the neuron weight vector, because the neuron weight and input vectors are normalized to a length of one. Therefore, passing a vec-

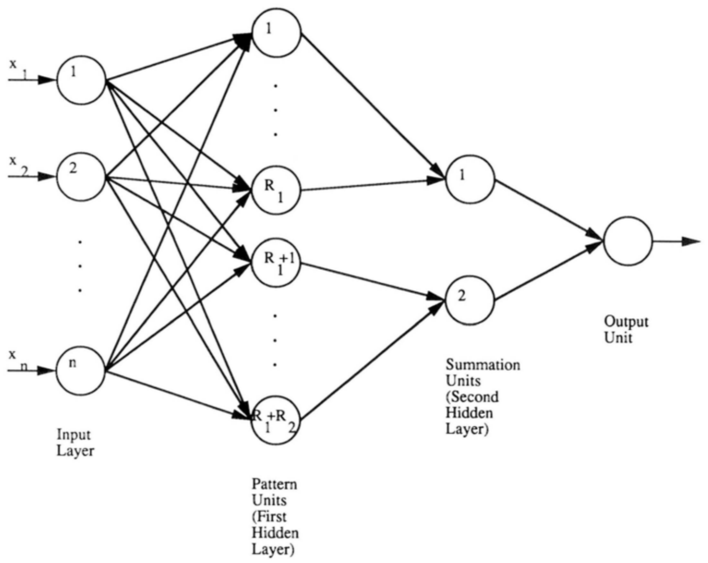
x1
x2
xn
1
2
n
Input Layer
1
R1
R1+1
R1+R2
Pattern Units (First Hidden Layer)
1
2
Summation Units (Second Hidden Layer)
Output Unit

FIGURE 6 Framework for a probabilistic neural network with n input nodes, and $R_1 + R_2$ pattern units.

tor of inputs, $\{x\}$, through the first hidden layer of neurons has the effect of constructing the components of Eq. (13) at the outputs of the neurons.

The purpose of the second hidden layer of neurons is to sum the elements output from the first hidden layer (the pattern units) corresponding to $\theta = \theta_A$ and $\theta = \theta_B$. (To see how this is done, refer again to Fig. 6. These units are called summation units in PNN terminology.) We do this by connecting the outputs of the first R_A hidden layer units to the first neuron in the second hidden layer, and setting all the corresponding weights in that hidden layer to one. We connect the outputs of the first hidden layer units indexed $R_A + 1$ through $R_A + R_B$ to the second neuron in the second hidden layer, and set all the corresponding weights to one. The effect of making the connections as described above is to set the complete set of PNN weights as follows.

$$w_{1k}^{(2)} = \begin{cases} 1, & k = 1, \ldots, R_A \\ 0, & k = R_A + 1, \ldots, R_A + R_B \end{cases}$$
$$w_{2k}^{(2)} = \begin{cases} 0, & k = 1, \ldots, R_A \\ 1, & k = R_A + 1, \ldots, R_A + R_B \end{cases}. \tag{17}$$

We simply wish to pass the net inputs through to the next layer of the PNN, so the activation function used on the two neurons in the second hidden layer is simply the identity.

Finally, a single output unit multiplies the output of the second neuron in the second hidden layer by a constant, then sums this with the output of the first neuron in the second hidden layer to form its net input. This is accomplished by setting the weight vector in the output neuron to $\{W^{(\text{out})}\} = (1, C)^T$, where

$$C = -\frac{h_B L_B}{h_A L_A}\frac{R_A}{R_B} \tag{18}$$

and computing the net input using Eq. (8). The activation function in the output unit is the binary function defined

$$f(\text{net}) = \begin{cases} -1, & \text{net} < 0 \\ 1, & \text{net} > 0 \end{cases}. \tag{19}$$

This accomplishes the comparison between the two sides of Eq. (12), and completes the operation of the PNN. When the output equals -1, then the input vector is judged to be a member of class B, and when the output equals 1, the input is judged to be a member of class A.

Though there appear to be many potential applications for PNNs in mechanical system assessment, few applications appear in the literature. One application is presented by Regelbrugge and Calalo (1992). They show how a PNN can be trained to identify the dynamic response of a structure. The input data used to train the PNN are the Fourier transforms of strains on the structure. The outputs sought from the PNN are decisions regarding whether or not a structural mode occurs within a band of frequencies. They use experimentally obtained data to show that when motion in a given mode appears in the training data, then the mode can be reliably identified by the PNN.

The main strength of the PNN is that it is easy to train: we simply normalize the training vectors and use them for the weights in the neurons of the first hidden layer. This is clearly a fast operation. The primary weakness of the PNN is that it yields results that are binary, and this framework may be difficult to employ in the solution of many mechanical system problems. This disadvantage may be overcome by the specification and simultaneous use of several PNNs in the solution of a problem. It does appear that many of the problems that have been analyzed using BPNs could more easily and efficiently be analyzed with PNNs, particularly those that require classification of system response.

Associative Memory

The associative memory (AM) is one of the most simply configured ANNs, yet it has substantial capability in specific applications. The AM is a very simple example of a network like the one shown in Fig. 1, and described in the section on general operation of ANNs. The AM has an input layer and an output layer only. It derives its name from its purpose, that is, it is an ANN with a memory defined by its weights, that associates vectors $\{x\}$ and $\{z\}$. In its various implementations, the AM is sometimes used with discrete data and sometimes with continuous data; it is sometimes used in a unidirectional and sometimes in a bidirectional mode. A unidirectional, linear AM will be described in the following.

A unidirectional, linear AM has an input layer whose nodes simply distribute the elements of the vector $\{x\}$ presented to them, to the output neurons. Net inputs to the output neurons are computed as usual, that is, using Eq. (8). The output of the neuron equals the net input, in a linear AM. Therefore, the output vector is simply the result of a linear matrix operation on the input vector.

Let $\{x_j\}$, $\{z_j\}, j = 1, \ldots, R$, denote the training set for an AM. An AM is said to be interpolative when it yields an output vector $\{y\}$ that is an interpolation of the vectors $\{z_j\}, j = 1, \ldots, R$, when presented with an input vector $\{x\}$ that comes from the space of vectors $\{x_j\}, j = 1, \ldots, R$. Interpolative AMs with continuous valued input and output vectors appear to be the ones with the greatest potential for effective use in the solution of problems in the analysis and design of mechanical systems.

Training of an AM depends on its function and rules of operation. One linear AM that can be very easily constructed relies on the assumption that the $\{x_j\}, j = 1, \ldots, R$, form an orthonormal basis, that is, $\{x_j\} \cdot \{x_k\} = \delta_{j-k}$, where δ_m is the Kronecker delta, equal to one when its subscript is zero, and zero otherwise. This AM has the following form.

$$\{y\} = \left[\sum_{j=1}^{R} \{y_j\}\{x_j\}^T\right]\{x\}. \qquad (20)$$

Because the $\{x_j\}, j = 1, \ldots, R$, are orthonormal, any vector $\{x\}$ with the appropriate dimension can be represented

$$\{x\} = \sum_{j=1}^{R} \gamma_j\{x_j\}. \qquad (21)$$

Again, because of the orthonormality of the $\{x_j\}$, $j = 1, \ldots, R$, the AM yields the output

$$\{y\} = \sum_{j=1}^{R} \gamma_j\{y_j\}. \qquad (22)$$

A situation more frequently encountered involves a training set $\{x_j\}$, $\{z_j\}, j = 1, \ldots, R$, where the $\{x_j\}, j = 1, \ldots, R$, do not form an orthonormal basis. In this case, we can still express the function of the AM with the relation

$$\{y\} = [W]\{x\}. \qquad (23)$$

However, we must establish the matrix $[W]$ via a training procedure. Potential approaches involve optimization via least squares, for example. The approach presented by Kalaba and Udwadia (1990, 1993) requires that the cost function defined

$$E = q\|\{W\}\{x\} - \{y\}\|^2 + (1 - q)\|[W]\|^2 \qquad (24)$$

be minimized with respect to $[W]$. Minimization of the first term on the right side improves the accuracy of the prediction of the output $\{y\}$ via Eq. (23). Minimization of the second term diminishes the potential for ill-conditioning in the matrix $[W]$. The quantity q determines the relative importance attached to accuracy of the prediction and conditioning of $[W]$. Because the matrix $[W]$ occurs as a quadratic in the expression of Eq. (24), the expression has a unique minimum that is governed by a linear equation. This can be solved to optimize the mapping in Eq. (23).

Other versions of the AM are also available. For example, some versions apply the sigmoidal activation function of Eq. (10) to the net inputs of the output layer neurons to produce the AM outputs. Introduction of this nonlinearity into AM increases its potential to model nonlinear system mappings. A version of the AM called the functional link is discussed by Hajela and Berke (1991). This ANN not only uses the actual inputs in the input vector to the AM, but it augments this with nonlinear functions of the actual inputs. This modification increases the ANN's potential to accurately represent nonlinear system mappings even more.

Kalaba and Udwadia (1990, 1993) use the approach to Eq. (23) and (24) to perform parameter identification in a dynamically excited, nonlinear mechanical system. The objective of their analysis is to estimate the parameters of a system, corresponding to a specific measured response. To commence an analysis, they generate mechanical system responses, $\{z_j\}, j = 1, \ldots, R$, using mechanical system parameters, $\{x_j\}$, $j = 1, \ldots, R$, chosen from the region where the parameters of the measured system are thought to reside. The training data can be obtained from numerical analysis, and need not be in the near vicinity of the true parameters. The training data are used to identify an AM following the procedure described above. The parameter vector corresponding to the measured response is estimated using the AM. Next, more training data from the vicinity of the newly created parameter estimate are generated, and these data are used to update and improve the AM weights. The AM weight matrix is recursively improved with each new data set, and the newly created weight matrices are used to make new estimates of system parameters. This computation cycle is repeated until convergence occurs in the parameter estimates. The authors used this technique to identify the parameters of some nonlinear systems, including one governed by the Van der Pol equation, a Lorenz attractor, and a Duffing oscillator. They obtained accurate results even when the system under consideration was highly nonlinear, and another technique diverged in its attempt at parameter identification.

A major strength of the linear AM is its simplicity of implementation and use. In fact, this general approach has been applied in many engineering analysis frameworks not referred to as ANNs. It is clearly applicable when the input–output relations governing a system are linear, but the study by Kalaba and Udwadia (1991, 1993) shows that it can be used in the nonlinear identification framework. Further, the introduction of nonlinear activation functions in the output neurons, and use of the ideas of the functional link discussed by Hajela and Berke (1991) greatly improve the potential applications of the AM. The primary weakness of the AM is that because of its simplicity it may have difficulty in simulating very nonlinear mappings, in some situations, unless a special sequence of operations is introduced to the analysis.

Radial Basis Function Network

The radial basis function network (RBFN) is an ANN that appears to combine some of the advantages of the BPN and the PNN to yield a tool that is capable of accurately mapping inputs to outputs, is relatively fast to train, and does not require (in its modified form) a large amount of training data. The concept of the RBFN is based upon ideas of local functional approximation, and it is different in form and operation from the ANNs described above. An early article on the subject was written by Moody and Darken (1989).

RBFNs can be trained faster than BPNs to perform similar tasks for several reasons. First, they involve local approximation of the influence of an input on the output, therefore, only the parameters in a part of the network need to be trained upon presentation of an exemplar. Second, an RBFN has only one hidden layer. Third, training of the RBFNs can be divided into linear and nonlinear parts. The nonlinear part is a self-organizing activity in which some parameters in the model are trained to represent the training input data set. The linear part trains the amplitudes in the local representations, and converges rapidly. Yet the RBFN, as first introduced, requires much more training data than a BPN to perform a particular task, and is subject to the same interpolation limitations.

A modified version of the RBFN was introduced by Jones et al. (1990) with the development of the connectionist normalized linear spline (CNLS) network. Introduction of normalization and an extra term in the local approximation appear to give it improved interpolation capabilities, and less stringent training requirements compared to the RBFN. We will describe, in the following, the CNLS.

First, note that although the CNLS can be characterized in the same graphical framework as the ANNs described above (Figs. 1, 2), it is more convenient to use another. This framework still uses the general geometry shown in Fig. 1; however, a measure of the inputs is computed and passed to all neurons in the hidden layer, and the functional operation of the neurons in the hidden layer is different from the sequence of operations shown in Fig. 2.

The operations performed in the neurons of the hidden layer in the CNLS can be motivated as follows. Recall that the purpose of an ANN is to develop an accurate approximation, $y = h(\{x\})$, to a mapping, $z = g(\{x\})$. (The scalar output case will be developed in the following, although multiple CNLSs can be used to approximate a vector output.) Consider a family of basis functions, $u(\{x\}, \{c_j\})$, $j = 1, \ldots, N$, each of which is localized in the region of a center, $\{c_j\}$, but none of which equals zero when evaluated at any finite valued vector, $\{x\}$. The basis functions are not necessarily orthogonal, nor uniformly spread, and do not all have the same widths. For example, a functional form that is widely used is,

$$u(\{x\}, \{c_j\}) = \exp(-\beta_j \|\{x\} - \{c_j\}\|^2) \quad (25)$$

the nonnormalized, multivariate normal pdf. The parameter β_j establishes the specific degree of localization of the jth basis function. Large values of β_j correspond to narrow widths in the basis functions, and small values of β_j correspond to large widths in the basis functions. The following identity involving the desired input–output relation, $z = g(\{x\})$, can be written in terms of the basis functions.

$$g(\{x\}) = \frac{\sum_{j=1}^{N} g(\{x\})u(\{x\}, \{c_j\})}{\sum_{j=1}^{N} u(\{x\}, \{c_j\})}. \tag{26}$$

The identity suggests that an approximation for the desired input–output relation can be established by replacing the expression for $g(\{x\})$ on the right side by the first two terms of its Taylor series expansion, $g_j + \{d_j\}^T(\{x\} - \{c_j\})$, where g_j is the function $g(.)$ evaluated at $\{c_j\}$, and $\{d_j\}$ is the vector of partial derivatives of $g(.)$ with respect to the elements in $\{x\}$, evaluated at $\{c_j\}$. The approximation is

$$h(\{x\}) = \frac{\sum_{j=1}^{N} [g_j + \{d_j\}^T(\{x\} - \{c_j\})]u(\{x\}, \{c_j\})}{\sum_{j=1}^{N} u(\{x\}, \{c_j\})}. \tag{27}$$

The CNLS implements such an approximation.

The functional relations described above are implemented in feedforward operation of the CNLS as follows. Before presentation of the input vector, $\{x\}$, to the hidden layer neurons, the denominator in Eq. (26) must be calculated. It is

$$\sum_{j=1}^{N} u(\{x\}, \{c_j\}). \tag{28}$$

(For consistency, this quantity can be presented to each of the neurons in the hidden layer of the CNLS as an element in the input, but the notation and description do not do so in the following.) The vector of inputs $\{x\}$ is presented to the jth neuron in the CNLS. This is first used to compute $g_j + \{d_j\}^T(\{x\} - \{c_j\})$, then it is used to compute $u(\{x\}, \{c_j\})$, based, for example, on Eq. (25). The output of the neuron is the product of these two quantities divided by Eq. (28). The outputs of all the neurons are transmitted to the output neuron, and it computes the net input in the usual way. The output of the output neuron equals its net input, as reflected in Eq. (27).

The objective of training in the CNLS is the specification of the parameters, g_j, $\{d_j\}$, $\{c_j\}$, and $\beta_j, j = 1, \ldots, N$, to minimize the error in representing $z = g(\{x\})$, the desired mapping, with $y = h(\{x\})$, the output of the CNLS. Note that the output of the CNLS is linear in g_j and $\{d_j\}$; therefore, if the other parameters are known, then these parameters can be optimized in closed form, or by using an iterative approach. Because of the linearity, training proceeds rapidly when done iteratively. The training approach presented in Jones et al. (1990) recommends that the parameters g_j, $\{d_j\}$, and $\beta_j, j = 1, \ldots, N$, be trained iteratively, and that the parameters $\{c_j\}$, $j = 1, \ldots, N$, be trained at discontinuous intervals using, for example, a genetic algorithm. The reason for use of the genetic algorithm in the training of the $\{c_j\}, j = 1, \ldots, N$, is that the error is a nonlinear function of these vectors.

The CNLS should provide an accurate approximation to $z = g(\{x\})$ when the number of basis functions is sufficient to represent the inputs to a system; the centers of the functions, $\{c_j\}, j = 1, \ldots, N$, are chosen to accurately represent the input data; the $\beta_j, j = 1, \ldots, N$, are chosen optimally; and the function $g(\{x\})$ does not vary rapidly compared to the widths of the basis functions.

A graphical sense for how this mapping approximation works can be obtained by reference to Figs. 5(a) and 5(b), because those mappings also involve the use of normal pdf curves. When the input to the CNLS is a bivariate vector, the CNLS provides a mapping from any input to a scalar output value, the ordinate in the plotted surfaces. Training of the CNLS involves the selection of the mapping parameters to optimally represent a desired relation, $z = g(\{x\})$. Training occurs over a region of the space defined by $\{x_j\}$, $j = 1, \ldots, n$.

Only one application of an RBFN was found in the literature. Cios, Vary, Berke, and Kautz (1992) use both a BPN and an RBFN to map spectral information obtained from acousto–ultrasonics tests to the elastic modulus of composite specimens. To perform their tests, the authors excite a composite specimen under tension using a wide band pulser. At another point on the specimen, they use an acoustic receiver to measure response time histories. These responses are used to estimate response spectral density. The response spectral densities at some frequencies (where the magnitude is greatest) are used as the training inputs. The elastic moduli obtained from tensile testing are used as the training output. Both a BPN and an RBFN were used to approximate the mapping. It was found that the RBFN was superior to the BPN in both training time and accuracy. In addition to a description of the specific study, the paper by Cios et al. (1992) also provides a brief yet clear description of some literature related to and useful suggestions for the use of RBFNs.

Though only one application of the RBFN or CNLS was found in the literature related to mechanical systems, the RBFN and CNLS are still considered worthy of discussion because they appear to provide very good alternatives to the BPN, and the BPN is widely used. Most importantly, the CNLS and RBFN can be trained rapidly. Moreover, they present alternatives to the BPN that appear to be unconditionally stable when used in the recurrent framework.

Recurrent Neural Networks

The recurrent neural network is not a totally separate form of ANN, but simply a framework for using an ANN like the BPN, or CNLS, in a manner that is meant to adapt its operation to the direct simulation of the step-by-step response of a system in time. Figure 7 shows an ANN structured in one form of a recurrent configuration. An ANN is said to be recurrent when the outputs of some or all its neurons are taken as inputs to some or all the neurons in the ANN, including the neuron that produced the output. The recurrent ANN shown in Fig. 7 clearly implements only a portion of the potential recurrent interconnections.

The effects that are simulated by making ANNs recurrent can be characterized in many ways, but a characterization of particular importance in mechanical system dynamics is that of autoregression (AR). When an ANN is trained to accept vector inputs, $\{x(t)\}$, that are a function of time, and a portion of the elements in $\{x(t)\}$ represent measures of excitation to a system, the remainder of the elements in $\{x(t)\}$ represent measures of system response at previous times, and the ANN output, $\{y(t)\}$, represents measures of system response as a function of time, then the mapping approximated in the ANN is the solution of the differential equations governing the response in the system of interest, over a time increment. This sort of simulation is actually ARMA in nature. When the time increment over which the simulation is executed is small enough, the AR portion of the input pattern will be similar to the output pattern, when the mechanical system under consideration displays continuous response. This simply means that measures of response do not vary rapidly during very short time intervals. The implication of all this is that the mapping performed in the ANN may be relatively simple and efficient to develop.

The step-by-step, temporal response of nonlinear mechanical systems can be simulated using the recurrent ANN. There is substantial advantage to the recurrent approach when

1. the relation between the overall excitation and system parameters and the response measures of interest is highly complex and nonlinear;
2. the expense of computing (using nonneural techniques) or experimentally simulating the system response is great; and
3. numerous system response simulations are required.

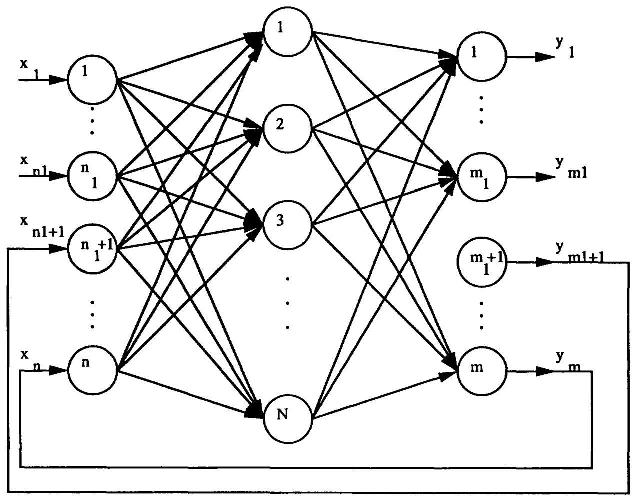

FIGURE 7 Framework for a recurrent neural network with n input nodes, N hidden level neurons, and m output neurons.

If a recurrent model of system behavior can be efficiently developed and economically run, then the recurrent ANN can be used to perform the required simulations. The results of these simulations might be used, in turn, to identify another ANN relating excitation and system parameters to important measures of system response.

Training of a recurrent network is similar to training of a nonrecurrent network. The exemplars, $\{x_j\}$, $\{z_j\}$, $j = 1, \ldots, R$, used to train a recurrent network are the inputs to and the states of the system to be modeled at time intervals during the response of the system. For example, in a particular training cycle, the input vector $\{x\}$ might be a combination of system excitation values and system responses at time $t - 1$, and the output vector $\{y\}$ might be system responses at time t. Training of recurrent ANNs is discussed, for example, in Williams and Zipser (1989).

Although there are advantages to the use of recurrent networks, there are also potential problems of which the user should be aware, the most important of which is that recurrent networks are not guaranteed to be stable in all implementations. For example, recurrent BPNs can become unstable. The hope is that if the system to be simulated is stable, and the ANN training is adequate, then the simulation will be stable and accurate. Of course, the stability of recurrent ANNs can be analyzed with the usual techniques for dynamic system stability analysis (see, for example, Parker and Chua, 1989).

Other Neural Network Applications

The articles briefly summarized in the previous sections show that there is a wide variety of applications for ANNs in mechanical system simulation, identification, and assessment. Applications that relate to structural dynamics and the corresponding ANN requirements can be summarized in matrix form as shown in Table 1. The ANN inputs that are required and the outputs that can be generated are listed for generic problems in structural dynamics. Based on the literature, it appears that only a small fraction of the potential applications of the various types of ANNs have been investigated; there are many models and applications yet to be studied.

A fundamental use of ANNs associated with mechanical system studies is the simulation of system response for arbitrary excitations. Many types of ANNs can be trained to simulate linear and nonlinear system response. The ANNs can be trained in applications where the system parameters either are or are not specified. A basic

Table 1. Matrix Defining Application Environments for ANNs

Problem	Inputs	Outputs
Mechanical system response simulation	Mechanical system excitation and parameters	Measures of mechanical system response at one point or multiple discrete points
Mechanical system parameter identification	Mechanical system excitations and responses	Parameters of mechanical system
Mechanical system, excitation identification	Mechanical system parameters and response	Mechanical system excitation
Mechanical system excitation and response assessment	All relevant measures of system condition to be assessed, including any or all of mechanical system excitation, response, and parameters	Assessment of system condition

Notes:

1. Excitation can be defined in terms of time history, Fourier transform of time history, or parameters in a parametric model. Excitation can be specified at one or more discrete points, or the parameters of a distributed excitation model can be output. On input, excitation need not be specified if it does not change.
2. Response can be specified as time history, Fourier transform of time history, or parameters in a parametric model of time history. On input, response need not be specified if it does not change. Output can be one or more measures of motion at time $t + \Delta t$, when input involves system motion at time t; it can also be a measure of motion-like peak motion over all time.
3. System parameters can be defined as multiple local parameters or parameters in a spatial parametric model. Parameters can refer to material or geometry, and can be either local system parameters or parameters in a global model. On input, system parameters need not be specified if they do not change.
4. Assessment of system condition can judge whether or not system fails, whether or not excitation and response satisfy some conditions, source of excitation or response, etc.

motivation for simulating system response is the desire to characterize features of the response excited by various excitations, in systems with various sets of structural parameters. ANNs provide a fast, accurate, and efficient means for simulating structural response, and this is especially important in Monte Carlo applications. Many Monte Carlo analyses have traditionally used finite element analysis to obtain the response of a (perhaps random) system to random excitation. In a Monte Carlo analysis performed on a fixed budget, introduction of ANNs into the analysis framework may permit an increase in accuracy resulting from an increase in the number of simulations run during prediction of a particular event. Direct (non-Monte Carlo) methods for the prediction of the probabilistic character of the response of a mechanical system also require a large number of response predictions, so the use of ANNs in this application can improve the efficiency of analysis.

An important advantage to the use of ANNs for system simulation is that both analytically and experimentally obtained data can be used in the training of ANNs. In fact, iterative techniques can be developed to identify the parameters of an analytical model based on experimental results (following, for example, the approach of Kalaba and Udwadia, 1990, 1993), then use the analytical model to generate data to augment the data base for training the ANN.

Parameter identification of mechanical systems in classical model frameworks can also be considered an area for independent analysis. When the analyst possesses both measured data from an experiment and an analytical model from which analytical predictions of system behavior can be obtained, then a parameter identification in the framework of the analytical model can be performed. A possible framework for performing this type of analysis is the one presented by Kalaba and Udwadia (1991, 1993); however, ANNs other than the AM might be used. An advantage of performing the system identification using ANNs is that nonlinear systems can be accommodated directly.

A broad area of applicability for ANNs involves the structural design and design optimization of mechanical systems subjected to static and dynamic environments. Some of the references have already introduced this topic for static structural optimization. Because numerous analyses of structural response are required in the design optimization of a mechanical system, the introduction of ANNs can substantially improve the efficiency of the design process. The increase in efficiency of the design process will be especially important in the design of mechanical systems that display nonlinear response.

The identification of mechanical system excitations is based on identification of some system parameters along with measurement of responses of the forced system. Such identifications are now usually performed using linear models and techniques, and, in fact, one method for force identification uses a linear combiner in the force prediction (Bateman, Mayes, and Carne, 1992). The linear combiner is simply a portion of the neuron model shown in Fig. 2, and is discussed in detail in Widrow and Stearns (1985). ANNs that generate continuous valued outputs appear to have good potential for reconstruction of input forces, particularly where the inputs excite nonlinear response in the structure upon which the measurements are made.

Health monitoring, damage assessment, and damage prediction applications can use ANNs for data analysis. Specifically, approaches that seek to assess the characteristics of structural members following major events can be implemented. Both direct and indirect measures of system component and overall system capability to sustain future loads can be developed. For example, estimates of member stiffnesses following application of a major excitation (as in Wu, Ghaboussi, and Garrett, 1992) can be developed, and they provide a direct measure of a mechanical system's load-carrying capability. Alternately, indirect measures of systems' load-carrying capability such as the accumulated damage in Miner's rule for fatigue in metals, can be estimated and used, in turn, to estimate component health. When the probability model governing future environments for a mechanical system is known, then ANNs like the ones described above can be used to predict the probability of survival of a structure.

PNNs can be used in the general health monitoring framework to make judgements about the response of a structure. In particular, in health monitoring and damage assessment applications we can build a framework for answering questions about the survival or failure of a mechanical system. The answers to such questions involve judgements based on measured response signals. In some sense, these investigations seek to develop inferences about the joint behavior of mechanical system parameters and the excitations that drive the systems whose response is measured.

CONCLUSION

Beyond the topic areas mentioned here, described generically in Table 1, and discussed in the literature, there are certainly other mechanical system applications that can be studied using ANNs. ANNs have been shown to be capable of accurate simulation, identification, and assessment of mechanical system behavior. Their efficiency of operation will certainly lead to applications and improvements in numerous areas of mechanical system design, analysis, and control.

This work was supported by the Department of Energy under Contract DE-AC04-76DP00789.

APPENDIX: SELECTED ABSTRACTS FROM ARTICLES ON NEURAL NETWORKS

Berke, Laszlo, and Hajela, Prahbat, 1992, "Applications of Artificial Neural Nets in Structural Mechanics," *Structural Optimization,* Vol. 4, pp. 90–98.

The fundamentals of neurologically motivated computing are briefly discussed. This is followed by presenting two examples of the many possible applications in structural mechanics. Both of these are oriented towards structural optimization. In the first example, a neural net model of the structural response is created and then attached to any conventional optimization algorithm. In the second, a neural net model of an experienced designer is created which is knowledgeable within a narrow class of structural concepts.

Cios, K. J., Vary, A., Berke L., and Kautz, H. E., 1992, "Application of Neural Networks to Prediction of Advanced Composite Structures Mechanical Response and Behavior," *Computing Systems in Engineering,* Vol. 3, pp. 539–544.

Two types of neural networks were used to evaluate acousto–ultrasonic data for material characterization and mechanical response prediction. The neural networks included a simple feedforward network (back propagation) and a radial basis functions network. Comparisons of results in terms of accuracy and training time are given. Acousto–ultrasonic (AU) measurements were performed on a series of tensile specimens composed of eight laminated layers of continuous, SiC fiber reinforced Ti-15-3 matrix. The frequency spectrum was dominated by frequencies of longitudinal wave resonance through the thickness of the specimen at the sending transducer. The magnitude of the frequency spectrum of the AU signal was used for calculating a stress–wave factor based on integrating the spectral distribution function and used for comparison with neural networks results.

Goh, C. J., and Noakes, Lyle, 1993, "Neural networks and identification of systems with unobserved states," *Journal of Dynamic Systems, Measurement and Control (ASME),* Vol. 115, pp. 196–203.

This article considers a nonlinear control system, whose structure is not known (apart from the order of the system) and whose states are not observed. The output of the system is observed for a period of time using persistently exciting input, and the observation is used to train a neural network emulator whose output approximates that of the original system. Such an explicit dynamical relationship between the input and the output is useful for the purpose of construction of an output feedback controller for nonlinear control systems. Specialization of the method to linear systems allows swift convergence and parameter identification in some cases.

Hajela, P., and Berke, Lazlo, 1992, "Neural Networks in Structural Analysis and Design: An Overview," *Computing Systems in Engineering,* Vol. 3, pp. 525–538.

Considerable recent interest has been shown in the application of neural networks to problems of structural analysis and design. This paper provides an overview of the state of the art in this emerging field, and a survey of the published applications in structural engineering. Such applications have included the use of neural networks in modeling nonlinear behavior of structures, as a rapid reanalysis capability in optimal design, and in developing problem-parameter sensitivity of optimal solutions for use in multilevel decomposition-based design. While most of the applications reported in the literature have been restricted to the use of the multilayer perception architecture and minor variations thereof, other network architectures have been successfully explored, including the adaptive resonance theory network, the counterpropagation network, and the Hopfield Tank model. Applications of these architectures have included solving inverse analysis problems, pattern matching in structural analysis and design, and as an optimization tool for generically difficult optimization problems, in particular, those characterized by being NP-complete.

Hajela, P., and Berke, Lazlo, 1991, "Neurobiological Computational Models in Structural Analysis and Design," *Computers and Structures,* Vol. 41, pp. 657–667.

This paper examines the role of neural computing strategies in structural analysis and design. A principal focus of the work resides in the use of neural networks to represent the force–displacement relationship in static structural analysis. Such models provide computationally efficient capabilities for reanalysis, and appear well suited for application in numerical optimum design. This paper presents an overview of the neural computing approach, with special emphasis on supervised learning techniques adopted in the present work. Special features of such learning strategies which have a direct bearing on numerical accuracy and efficiency, are examined in the context of representative structural optimization problems.

Holland, John, 1992. "Genetic Algorithms," *Scientific American,* July.

This paper describes computer programs that "evolve" in ways that resemble the natural selection of living organisms. Natural selection eliminates two of the greatest hurdles in software design: advance specification of the features of a problem and the actions a program should take to deal with them. By employing the genetic mechanism of natural selection, researchers may be able to "breed" programs that solve problems even when no one can fully understand their structures.

Hornik, Kurt, Stincombe, Maxwell, and White, Halbert, 1989, "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks,* Vol. 2, pp. 359–366.

This paper rigorously establishes that standard multilayer feedforward networks, with as few as one hidden layer using arbitrary squashing functions, are capable of approximating any Borel measureable function from one finite dimensional space to another, to any desired degree of accuracy, provided many hidden units are available. In this case, multilayer feedforward networks are a class of universal approximators.

Jones, R. D., Lee, Y. C., Qian, S., et al. 1990, "Nonlinear Adaptive Networks: A Little Theory, A Few Applications," *Cognitive Modeling in System Control,* June.

This paper presents the theory of nonlinear adaptive networks and discusses a few applications—in particular, the theory of feedforward backpropagation networks. The following theories are also discussed: Connectionist Normalized Linear Spline network in both its feedforward and iterated modes; stochastic cellular automata; applications to chaotic time series, tidal prediction in Venice Lagoon, sonar transient detection, control of nonlinear processes, balancing a double inverted pendulum, and design advice for free electron lasers.

Kalaba, Robert, and Udwadia, Firdaus, 1991, "An Adaptive Learning Approach to the Identification of Structural and Mechanical Systems," *Computer Mathematics Applications,* Vol. 22, pp. 67–75.

The identification of parameters in models of structural and mechanical systems is an important problem. The usual approaches are successive approximation schemes which require good initial guesses for rapid convergence. This paper shows how such initial approximations may be obtained. Notions from the field of artificial neural networks are used. In fact, new adaptive schemes for learning are presented and used in parameter estimation for both linear and nonlinear systems.

Kalaba, Robert, and Udwadia, Firdaus, 1993. "Associative Memory Approach to the Identification of Structural and Mechanical Systems," *Journal of Optimization Theory and Applications,* Vol. 76, pp. 207–223.

This paper presents a new method for identification of parameters in nonlinear structural and mechanical systems in which the initial guesses of the unknown parameter vectors may be far from their true values. The method uses notions from the field of artificial neural nets and, using an initial set of training parameter vectors, generates in an adaptive fashion other relevant training vectors to identify the parameter vector in a recursive fashion. The simplicity and power of the technique are illustrated by considering three highly nonlinear systems. The technique presented here yields excellent estimates with only a limited amount of response data, even when each element of the set comprising the initial training parameter vectors is far from its true value—in fact, sufficiently far that the usual recursive identification schemes fail to converge.

Larimore, Wallace, 1983, "System Identification, Reduced-Order Filtering and Modeling via Canonical Variate Analysis," *Proceedings of the 1983 American Control Conference,* H. Rao and P. Dorato, Eds., pp. 445–451, IEEE Service Center, Piscataway, NJ.

Very general reduced-order filtering and modeling problems are phased in terms of choosing a state based on past information to optimally predict the future as measured by a quadratic prediction error criterion. The canonical variate method is extended to approximately solve this problem and give a near optimal, reduced-order state model. The approach is related to the Hankel norm approximation method. The central step in the computation involves a singular value decomposition which is numerically very accurate and stable. An application to reduced-order modeling of transfer functions for stream flow dynamics is given.

McAulay, A. D., 1988, "Optical Neural Network for Engineering Design," *Proceedings of the IEEE 1988 National Aerospace and Electronics Conference,* Vol. 4, pp. 1302–1306.

An optical neural-network architecture is proposed that captures engineering design expertise and makes it available to designers. The network and its use as an associative memory are described. A novel, fast learning algorithm is shown to be orders-of-magnitude faster than backpropagation. Reasons are presented for the feasibility of an optical implementation that uses novel optical devices. A structural example illustrates how engineering design expertise is captured and recovered from the network.

Moody, J., and Darken, C. J., 1989, "Fast Learning in Networks of Locally-Tuned Processing Units," *Neural Computation,* Vol. 1, pp. 281–294.

The authors propose a network architecture which uses a single internal layer of locally tuned processing units to learn both classification tasks and real-valued function approximations. They consider training such networks in a completely supervised manner, but abandon this approach in favor of a more computationally efficient hybrid learning method which combines self-organized and supervised learning. Their networks learn faster than backpropagation for two reasons: the local representations ensure that only a few units respond to any given input, thus reducing computational overhead, and the hybrid learning rules are linear rather than nonlinear, thus leading to faster convergence. Unlike many existing methods for data analysis, the network architecture and learning rules are truly adaptive and are thus appropriate for real-time use.

Parzen, Emmanuel, 1962, "On Estimation of a Probability Density Function and Mode," *Annals of Mathematical Statistics,* Vol. 33, pp. 1065–1076.

This paper discusses the method for construction of a family of estimates of the probability density function $f(x)$ and of the mode, which are consistent and asymptotically normal. The problem of estimation of the function and determination of the mode are discussed.

Regelbrugge, Marc, and Calalo, Ruel, 1992, "Probabalistic Neural Network Approaches for Autonomous Identification of Structural Dynamics," *Journal of Intelligent Material Systems and Structures,* Vol. 3, pp. 572–584.

This paper presents an application of Probabalistic Neural Networks (PNN) to identify dynamic responses of structures. The PNN architecture is described, and its operation relative to other neural network types is discussed. The instant learning property of the PNN is shown to be a critical advantage for applications where both adaptivity and autonomous operation are required. An example, PNN-based identification of structural responses from structural-member strain measurements is presented to illustrate operation of the network. The network is shown to be capable of classifying dynamics in a spatial–frequency domain very quickly using a small number of active elements.

Riva, Alberto, and Giorcelli, Ermanno, 1992, "Dynamic System Identification by Means of Neural Networks," *Proceedings of the 10th International Modal Analysis Conference,* pp. 928–933.

Neural nets have already demonstrated their capabilities for pattern recognition, control system and other identification processes. Their application developed in this work concerns the neural nets behavior when dealing with time domain data from dynamic systems. A FORTRAN routine simulating a backprop algorithm has been written in a general form to allow any configuration set, within a reasonable size, to treat input–output digital signals of linear and nonlinear systems. An overview of this algorithm is presented in the paper with some useful remarks concerning its relationships with signal processing. Two examples of parameter identification are presented and relationships with ARMA models highlighted; also mentioned is how more-refined training procedures can avoid local minima and improve parameter estimation. The examples are straightforward computer parameters of mechanical systems under test with a satisfying precision; nonlinear parameters are also derived.

Specht, Donald F., 1990, "Probabilistic Neural Networks," *Neural Networks,* Vol. 3, pp. 109, 118.

By replacing the sigmoid activation function often used in neural networks with an exponential function, a probabilistic neural network (PNN) that can compute nonlinear decision boundaries which approach the Bayes optimal is formed. Alternate activation functions having similar properties are also discussed. A four-layer neural network of the type proposed can map any input pattern to any number of classifications. The decision boundaries can be modified in real-time using artificial hardware 'neurons' that operate entirely in parallel. Provision is also made for estimating the probability and reliability of a classification as well as making the decision. The technique offers a tremendous speed advantage for problems in which the incremental adaptation time of back propagation is a significant fraction of the total computation time. For one application, the PNN paradigm was 200,000 times faster than back-propagation.

Tsutsumi, K., Katayama, K., and Matsumoto, H., 1988, "Neural Computation for Controlling the Configuration of 2-Dimensional Truss Structure," *1988 IEEE Conference on Neural Networks,* Vol. 2, pp. 575–586.

A method for manipulator position control was previously proposed by Tsutsumi et al. (1987) on the basis of the Hopfield scheme for neural computation. According to the proposed method, the manipulator is modeled not by using polar coordinates but by using the distances between the joint locations alone. The analog neural network actualizes parallel control by minimizing the total of the energy functions based on the distances, and it can easily treat not only the ordinary manipulator with rigid links but also the elastic arm. The proposed control method with parallelism and flexibility is further applied to a two-dimensional truss structure composed of elastic members. Simulation demonstrates that the configuration of the structure can be controlled by adjusting the initial configuration and by tuning the energy balance.

Vanluchene, R. D., and Sun, Roufei, "Neural Networks in Structural Engineering," *Microcomputers in Civil Engineering,* Vol. 5, pp. 207–215.

In the past few years literature on computational civil engineering has concentrated primarily on artificial intelligence (AI) applications involving expert system technology. This article discusses a different AI approach involving neural networks. Unlike their expert system counterparts, neural networks can be trained based on observed information. These systems exhibit a learning and memory capability similar to that of the human brain, a fact due to their simplified modeling of the brain's biological function. This article presents an introduction to neural network technology as it applies to structural engineering applications. Differing network types are discussed and a back-propagation learning algorithm is presented. The article concludes with a demonstration of the potential of the neural network approach. The demonstrations involves three structural engineering problems. The first involves pattern recognition; the second, a simple concrete beam design; and the third, a rectangular plate analysis. The pattern recognition problem demonstrates a solution which would otherwise be difficult to code in a conventional problem. The concrete beam problem indicates that typical design decisions can be made by neural networks. The last problem demonstrates that numerically complex solutions can be estimated almost instantaneously with a neural network.

Wang, Gou-Jen, Miu, Denny K., 1991, "Unsupervised Adaptive Neural-Network Control of Complex Mechanical Systems," *Proceedings of the 1991 American Control Conference,* pp. 28–29.

Unsupervised adaptive control strategies based on neural networks are presented. The tasks are performed by two independent networks which act as the plant identifier and the system controller. A learning algorithm using information embedded in the identifier to modify the action of the controller has been developed. Simulation results are presented showing that this system can learn to stabilize a difficult benchmark control problem, the inverted pendulum, without requiring any external supervision.

Werbos, Paul, 1989, "Neural Networks for Control and System Identification," *Proceedings of the IEEE Conference on Decision and Control,* pp. 260–265.

This paper first reviews the field of neuroengineering as a whole, highlighting the importance of neurocontrol (in robotics, in particular) and a few areas for future research. It will conclude with a few comments on neuroidentification. Neurocontrol is not an alternative to the broad disciplines of control theory and decision analysis. In fact, it may be understood as an extension or development of those fields to deal with a family of large, sticky problems which tend to require approximations and experiments rather than exact solutions and iron-clad guarantees of success. Control theorists have much to contribute to this emerging field.

Williams, R. J., and Zipser, D., 1989, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks," *Neural Computation,* Vol. 1, pp. 270–280.

The exact form of a gradient-following learning algorithm for completely recurrent networks running in continually sampled time is derived and used as the

basis for practical algorithms for temporal supervised learning tasks. These algorithms have (1) the advantage that they do not require a precisely defined training interval, operating while the network runs; and (2) the disadvantage that they require nonlocal communication in the network being trained and are computationally expensive. These algorithms allow networks having recurrent connections to learn complex tasks that require retention of information over time periods having either fixed or indefinite length.

Wu, X., Ghaboussi, J., and Garrett, J. H., Jr., 1992, "Use of Neural Networks in Detection of Structural Damage," *Computers and Structures,* Vol. 42, pp. 649–659.

Damage to structures may be caused as a result of normal operations, accidents, deterioration or severe natural events such as earthquakes and storms. Most often the extent and the location of the damage can be determined through visual inspection. However, in some cases visual inspection may not be feasible. The study reported in this paper is the first stage of a research project aimed at developing automatic monitoring methods for detection of structural damage. In this feasibility study we have explored the use of the self-organization and learning capabilities of neural networks in structural damage assessment. The basic strategy is to train a neural network to recognize the behavior of the undamaged structure as well as the behavior of the structure with various possible damage states. When the trained network is given the measurements of the structural response, it should be able to detect any existing damage. We have tried this basic idea on a simple structure and the results are promising.

Xirouchakis, Paul C., And Norris, Eugene M., 1991, "Column Buckling Mode Classification Using a Back Propagation Neural Network," *Proceedings of the 10th Conference on Electronic Computations,* ASCE, pp. 295–302.

A back propagation neural network is used to analyze and classify the buckling mode shapes of structural columns. The input patterns consist of vectors of digitized nondimensional deflection points of the buckling mode shapes of columns. The output consists of the mode number and the associated effective column length ratio as predicted by the network. The results of this study are of interest in the area of numerical finite element post-processing and results interpretation of structural mechanics applications. In particular, the objective of this paper is to investigate the column buckling mode and associated compressive buckling load classification potential of a back propagation neural network.

Yamamoto, K., 1992, "Modeling of Hysteretic Behavior with Neural Network and Its Application to Nonlinear Dynamic Response Analysis," *Applications of Artificial Intelligence in Engineering, Proceedings of the 7th International Conference,* pp. 475–486.

The neural networks developed by researchers in connectionism (a part of the field of artificial intelligence), are becoming very popular for solving pattern recognization problems. Neural networks already have been applied not only to qualitative pattern recognition, but also to quantitative approximation. There are a few attempts to apply this approach to structural analysis. In this paper, a backpropagation neural network was developed to represent hysteretic behavior, and the network was applied to nonlinear dynamic response analysis of a single-degree-of-freedom (SDOF) system.

REFERENCES

Bateman, V. I., Mayes, R. L., and Carne, T. G., 1992, "A Comparison of Force Reconstruction Methods for a Lumped Mass Beam," *Proceedings of the 63rd Shock and Vibration Symposium,* pp. 757–767.

Berke, L., and Hajela, P., 1992, "Applications of Artificial Neural Nets in Structural Mechanics," *Structural Optimization,* Vol. 4, pp. 90–98.

Cacoullos, T., 1966, "Estimation of Multivariate Density," *Annals of the Institute of Statistical Mathematics,* Vol. 18, pp. 179–189.

Cios, K. J., Vary, A., Berke, L., and Kautz, H. E., 1992, "Application of Neural Networks to Prediction of Advanced Composite Structures Mechanical Response and Behavior," *Computing Systems in Engineering,* Vol. 3, pp. 539–544.

Freeman, J. A., and Skapura, D. M., 1991, *Neural Networks, Algorithms, Applications, and Programming Techniques,* Addison–Wesley Publishing Company, Reading, MA.

Goh, C. J., and Noakes, L., 1993, "Neural Networks and Identification of Systems with Unobserved States," *Journal of Dynamic Systems, Measurement, and Control (ASME),* Vol. 115, pp. 196–203.

Hajela, P., and Berke, L., 1991, "Neurobiological Computational Models in Structural Analysis and Design," *Computers and Structures,* Vol. 41, pp. 657–667.

Hajela, P., and Berke, L., 1992, "Neural Networks in Structural Analysis and Design: An Overview," *Computing Systems in Engineering,* Vol. 3, pp. 525–538.

Holland, J. H., 1992, "Genetic Algorithms," *Scientific American,* Vol. 267, No. 1, pp. 66–72.

Hornik, K., Stinchcombe, M., and White, H., 1989, "Multilayer Feedforward Networks Are Universal Approximators," *Neural Networks,* Vol. 2, pp. 359–366.

Jones, R. D., Lee, Y. C., Qian, S., et al., 1990, "Nonlinear Adaptive Networks: A Little Theory, A Few Applications," *Cognitive Modeling in System Control,* The Santa Fe Institute, Santa Fe, NM.

Kalaba, R. E., and Udwadia, F. E., 1991, "An Adaptive Learning Approach to the Identification of Structural and Mechanical Systems," *Computer Mathematical Applications,* Vol. 22, pp. 67–75.

Kalaba, R. E., and Udwadia, F. E., 1993, "Associative Memory Approach to the Identification of Structural and Mechanical Systems," *Journal of Optimization Theory and Applications,* Vol. 76, pp. 207–223.

Larimore, W. E., 1983, "System Identification, Reduced-Order Filtering and Modeling Via Canonical Variate Analysis," *Proceedings of the 1983 American Control Conference,* H. Rao and P. Dorato, Eds., pp. 445–451, IEEE Service Center, Piscataway, NJ.

Moody, J., and Darken, C. J., 1989, "Fast Learning in Networks of Locally Tuned Processing Units," *Neural Computation,* Vol. 1, pp. 281–294.

Parker, T. S., and Chua, L. O., 1989, *Practical Numerical Algorithms for Chaotic Systems,* Springer–Verlag, New York.

Parzen, E., 1962, "On Estimation of a Probability Density Function and Mode," *Annals of Mathematical Statistics,* Vol. 33, pp. 1065–1076.

Press, W. H., Flannery, B. P., Teukolsky, S. A., Vetterling, W. T., 1988, *Numerical Recipes in C. The Art of Scientific Computing.* Cambridge University Press, Cambridge.

Regelbrugge, M. E., and Calalo, R., 1992, "Probabilistic Neural Network Approaches for Autonomous Identification of Structural Dynamics," *Journal of Intelligent Material Systems and Structures,* Vol. 3, pp. 572–584.

Riva, A., and Giorcelli, E., 1992, "Dynamic System Identification by Means of Neural Networks," *Proceedings of the 10th International Modal Analysis Conference,* pp. 928–933.

Rumelhart, D. E., Hinton, G. E., and McClelland, J. L., 1986, "A General Framework for Parallel Distributed Processing," Chapter 2 in *Parallel Distributed Processing, Explorations in the Microstructure of Cognition, Volume 1: Foundations,* MIT Press, Cambridge, MA.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J., 1986, "Learning Internal Representations by Error Propagation," Chapter 8 in *Parallel Distributed Processing, Explorations in the Microstructure of Cognition, Volume 1: Foundations,* MIT Press, Cambridge, MA.

Specht, D. F., 1990, "Probabilistic Neural Networks," *Neural Networks,* Vol. 3, pp. 109, 118.

Vanluchene, R. D., and Sun, R., 1990, "Neural Networks in Structural Engineering," *Microcomputers in Civil Engineering,* Vol. 5, pp. 207–215.

Wang, G.-J., and Miu, D. K., 1991, "Unsupervised Adaptive Neural Network Control of Complex Mechanical Systems," *Proceedings of the 1991 American Control Conference,* pp. 28–29.

Werbos, P. J., 1989, "Neural Networks for Control and System Identification," *Proceedings of the IEEE Conference on Decision and Control,* pp. 260–265.

Widrow, B., and Stearns, S. D., 1985, *Adaptive Signal Processing,* Prentice–Hall, New York.

Williams, R. J., and Zipser, D., 1989, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks," *Neural Computation,* Vol. 1, pp. 270–280.

Wu, X., Ghaboussi, J., and Garrett, J. H., 1992, "Use of Neural Networks in Detection of Structural Damage," *Computers and Structures,* Vol. 42, pp. 649–659.

Xirouchakis, P. C., and Norris, E. M., 1991, "Column Buckling Mode Classification Using a Back Propagation Neural Network," *Proceedings of the 10th Conference on Electronic Computations,* ASCE, pp. 295–302.

Yamamoto, K., 1992, "Modeling of Hysteretic Behavior with Neural Networks and Its Application to Nonlinear Dynamic Response Analysis," *Applications of Artificial Intelligence in Engineering. Proceedings of the 7th International Conference,* pp. 475–486.