SAND96-1861C
CONF-970113--2

USE OF ARTIFICIAL NEURAL NETWORKS FOR ANALYSIS OF COMPLEX PHYSICAL SYSTEMS

Allan Benjamin
Brad Altman
Risk Assessment Department

Chris O'Gorman
Ronald Rodeman
Thomas L. Paez
Experimental Structural Dynamics Department
Sandia National laboratories
Albuquerque, New Mexico

Abstract

Mathematical models of physical systems are used, among other purposes, to improve our understanding of the behavior of physical systems, predict physical system response, and control the responses of systems. Phenomenological models are frequently used to simulate system behavior, but an alternative is available - the artificial neural network (ANN). The ANN is an inductive, or data-based model for the simulation of input/output mappings. The ANN can be used in numerous frameworks to simulate physical system behavior. ANNs require training data to learn patterns of input/output behavior, and once trained, they can be used to simulate system behavior within the space where they were trained. They do this by interpolating specified inputs among the training inputs to yield outputs that are interpolations of training outputs. The reason for using ANNs for the simulation of system response is that they provide accurate approximations of system behavior and are typically much more efficient than phenomenological models. This efficiency is very important in situations where multiple response computations are required, as in, for example, Monte Carlo analysis of probabilistic system response.

This paper describes two frameworks in which we have used ANNs to good advantage in the approximate simulation of the behavior of physical system response. These frameworks are the non-recurrent and recurrent frameworks. It is assumed in these applications that physical experiments have been performed to obtain data characterizing the behavior of a system, or that an accurate finite element model has been run to establish system response. The paper provides brief discussions on the operation of ANNs, the operation of

two different types of mechanical systems, and approaches to the solution of some special problems that occur in connection with ANN simulation of physical system response. Numerical examples are presented to demonstrate system simulation with ANNs.

Introduction

Those trained in traditional methods for system analysis typically approach problems of determining responses from inputs by synthesizing input/output models based on first principles. Systems of interest might be physical, economic, or any general input/output relations. These approaches and their corresponding models are often referred to as phenomenological; the models are obtained by a deductive process. However, the human approach to learning to perform an activity, where for a given stimulus a desirable response is sought, is based on an inductive approach. For example, when a person wishes to learn to play tennis, he or she could study solid dynamics, fluid mechanics, physiology, etc., in an effort to deduce the fundamental and optimal techniques. But people do not usually learn to play this way. Rather, they learn through observation, experience, feedback. To learn to play tennis an individual might observe others, try playing the game, execute the actions required in the game, and modify these actions to approach the desired outcomes as closely as possible. After learning the fundamentals and training for a time, the beginning tennis player might test the skills he or she has learned. The typical person learning to play tennis does not understand the mathematics governing the game, and therefore, does not base play upon a phenomenological mathematical model. Rather, he or she learns by training. This and similar examples lead us to consider the modeling of complex systems and processes using an inductive framework.

An ANN is a tool used for the inductive modeling of system behavior. Its purpose is to simulate the behavior of another system following a period of training. A general discussion of several types of artificial neural networks is presented in Paez (1993). Some discussions of the step-by-step modeling of dynamic systems using ANNs are presented in Paez (1994, 1995, 1996).

ANNs were historically developed to simulate aspects of brain behavior and function using simple, yet realistic, elements. Discussions of the historical development of ANNs are given, for example, in Freeman and Skapura (1991) and Vanluchene and Sun (1990).

The basic idea behind simulation with an ANN can be depicted graphically, using the block diagram in Fig. 1. First, when a stimulation vector $\{x\}$ is input to an actual system, the system operates on the input to produce the output, $\{z\}$. When the same vector $\{x\}$ is input to an ANN the ANN operates on the input to produce the output $\{y\}$. The difference between $\{z\}$ and $\{y\}$ is the error $\{\varepsilon\}$. When we wish to create an ANN that simulates the actual system, we seek to adjust the parameters of the ANN so that when both the system and the ANN are presented with the same input, the output of the ANN differs from the output of the real system by some small amount, i.e., $\{\varepsilon\}$ is minimal in some sense. The ANN shown here is a general, nonrecurrent ANN, that is, it relates a general input to a general output. (The recurrent ANN will be described later.)

The objective of an ANN can be expressed mathematically using the following input/output notation and terminology. Let $\{x\}$ represent a vector of inputs presented to a system. The system to be simulated operates on the data in $\{x\}$, yielding an output vector $\{z\}=g(\{x\})$. The function g(.) is assumed deterministic but unknown. An ANN is constructed so that it can also be used to operate on the inputs $\{x\}$ to yield $\{y\}=h(\{x\})$. The function $h(.)$ is deterministic, has a preestablished framework, and parameters $\{p\}$. Given a sequence of exemplars $\{xj\}$, $\{zj\}$, j=1,...,R, (examples of correct input/output behavior of the actual system), we seek to adjust the parameters $\{p\}$ of the ANN to minimize the error between the actual system output $\{z\}$ and the ANN output $\{y\}$, when presented with the same input $\{x\}$. This is accomplished through a training process that involves error minimization through variation of the parameters $\{p\}$. The error minimization can be performed using a local search (for example, gradient search, least mean square, Newton's method) or global search (for example, simulated annealing, genetic algorithm). Once training is accomplished, it is hoped that, given an input $\{x\}$ different from those used during training, an ANN will yield outputs $\{y\}$, that are accurate approximations of the outputs $\{z\}$ produced by the system being modeled. The ANN serves as an interpolator of output vectors among the output exemplars as a function of the position of the input vector among the input exemplars.

The recurrent ANN studied here is a special type of ANN that recirculates some or all of the outputs of the ANN back to the input nodes. Figure 2 shows a block diagram of a recurrent ANN. To develop a recurrent ANN that simulates structural system behavior on a step-by-step basis, we may wish to divide ANN inputs into dependent and independent inputs. The independent inputs to the ANN, denoted $\{x_j\}$ in Fig. 2, may be quantities like forcing functions, boundary conditions, or material parameters at a particular time. The dependent inputs, denoted $\{y_j\}$ in Fig. 2, may be quantities like system response measures at the same time. The outputs of the recurrent ANN, denoted $\{y_{j+1}\}$ in Fig. 2, may be quantities like system response measures at a later time. In the recurrent framework, the outputs of a particular cycle of ANN computation are used as the dependent inputs during the next cycle of ANN computation.

This paper summarizes the results of some studies in which the responses of mechanical systems were simulated using a connectionist normalized linear spline (CNLS) network. This is a specific type of radial basis function network. The basic radial basis function network is described in Moody and Darken (1989), and the CNLS network is described in Jones, et.al., (1990). The CNLS network is described in the following section. Next, some techniques for reducing the information in system inputs and responses to a small set of characteristics are described. Following that, the general framework used for system response definition and system simulation are described. Finally, the results of some numerical examples are summarized, and conclusions are given.

The CNLS Network

Consider the following simple problem. We seek to develop an approximation to the jagged (or noisy) function in Figure 3. A means for accomplishing this is to develop a representation that is a sum of simple functions. For example, we can add three functions that are constants times normal probability density functions (the dashed curves) to form the smooth solid line. The smooth solid line is an approximation to the function we wish to represent. The CNLS net develops approximations in precisely this way, except that its coefficients are linear functions instead of constants, we use multivariate normal density functions instead of univariate normal density functions, and a normalization is included.

The CNLS net is motivated by the following identity. Let $z=g(\{x\})$ denote a mapping from the input vector $\{x\}$ to the output (scalar, in this case) z. Let us define the sequence of radial basis functions $u(\{x\},[\beta],\{c_j\})$, j=1,...,N, (the multivariate normal density functions to be specified, later) as scalar functions of the vector $\{x\}$, with parameters $\{c_j\}$, j=1,...,N, and $[\beta]$. The functions $u(\{x\},[\beta],\{c_j\})$, j=1,...,N, are nonzero for finite $\|x\|$, and will be described later. The following is an identity.

$$g(\{x\})\sum_{j=1}^{N}u(\{x\},[\beta],\{c_j\}) \equiv \sum_{j=1}^{N}g(\{x\})u(\{x\},[\beta],\{c_j\}) \tag{1}$$

This can be written because $g(\{x\})$ is not a function of the index j. Now approximate $g(\{x\})$ in the sum on the right hand side with the first two terms in its Taylor series, expanded about $\{c_j\}$. This is

$$g(\{x\}) \cong g_j + \{d_j\}^T\left(\{x\}-\{c_j\}\right) \tag{2}$$

where $g_j=g(\{c_j\})$ is a scalar constant, and the ith element of the vector $\{d_j\}$ is $\partial g/\partial x_i$ evaluated at $\{x\}=\{c_j\}$. The g_j and $\{d_j\}$ are the constant and linear terms, respectively, of the Taylor series approximation. Move the sum on the left hand side of Eq. (1) to the right hand side, and use Eq. (2) in Eq. (1). The result is the approximation

$$z \cong y = h(\{x\}) = \frac{\sum_{j=1}^{N}\left[g_j + \{d_j\}^T(\{x\}-\{c_j\})\right]u(\{x\},[\beta],\{c_j\})}{\sum_{j=1}^{N} u(\{x\},[\beta],\{c_j\})} \tag{3}$$

The operations on the right hand side, above, express the framework for the CNLS net approximation to $z=g(\{x\})$. The scalar y is the output of the CNLS net, and we seek parameters for the network that will make y a good approximation to z, the output of the actual system.

We now define the form of the radial basis function. It is typically taken to have the form of the multivariate normal probability density function, namely,

$$u(\{x\},[\beta],\{c_j\}) = \exp\left(-(\{x\}-\{c_j\})^T[\beta](\{x\}-\{c_j\})\right) \tag{4}$$

This is a "bell-shaped curve" defined on the n-dimensional real space. The center of the curve is defined by $\{c_j\}$, and the "width" of the curve in each of the directions of the elements of $\{x\}$ is an element in the diagonal matrix $[\beta]$. In practical applications, the parameters of the radial basis functions, $[\beta]$ and $\{c_j\}$, and the linear parameters, g_j and $\{d_j\}$, can be adjusted to make the surface approximate any desired shape. Clearly, the number of radial basis functions used in the approximation affects the accuracy that is achievable in the approximation.

The output y of the CNLS net, expressed in Eq. (3), is a linear function of g_j and $\{d_j\}$, and it is a nonlinear function of $\{c_j\}$ and $[\beta]$. Adjusting g_j and $\{d_j\}$ modifies the magnitude and derivatives of the CNLS representation of an output. Adjusting $\{c_j\}$ and $[\beta]$ modifies the locations where the approximation is centered, and the width of the basis functions. Training of the CNLS net involves the adjustment of all the parameters to optimize an input/output approximation. This training is accomplished using an approach that takes advantage of the combined linear plus nonlinear dependence on the parameters.

It is assumed that exemplars are available to train the CNLS net. Denote these $\{x_j\}$, z_j, j=1,...,R. The CNLS net is trained by presenting exemplars to the net, one set at a time. To start, the linear terms are chosen at random. During each and every training loop the linear terms are trained using the least mean square (LMS) algorithm, described, for example, in Widrow and Stearns (1985). The training modifies the linear terms to diminish the differences between the CNLS net output and the output exemplar z_j. The terms in $[\beta]$ are initiated as deterministic values. The mean square errors associated with particular b values are accumulated, and periodically, during training the β values are modified based on a quadratic approximation of the error surface, that minimizes the error. Finally, the centers $\{c_j\}$ are chosen using a stochastic search. The centers are chosen at random from among the input exemplars, $\{x_j\}$, j=1,...,R.. Training of the CNLS net is executed over a preestablished number of steps, and the mean square error is noted. The center selection process is repeated several times and the CNLS training process is restarted each time. The centers that yield the minimum error are then used in extended training of the CNLS net. Extended training of the CNLS net consists of the execution of one or more training epochs. In a given training epoch, we select some or all of the exemplars, $\{x_j\}$, z_j, j=1,...,R, randomize their order, and execute the training steps described above. We frequently use five to ten training epochs to select the optimal CNLS net centers, c_j, j=1,...,N. Extended training of the CNLS net often converges to an optimum in three to ten training epochs.

Data Reduction

The ANN modeling of physical systems can often be made more efficient and accurate by preprocessing the training data using any of a number of simplifying transformations. Among these are: (1) principal component analysis, (2) modal filtering, (3) elimination of statistically dependent components of motion, and (4) transformation of the components of motion to statistically independent, standard normal random signals. The hopes in using these transformations are that the ANN required to model a component of behavior will be simpler than a model for the entire system, and that a simpler model will be easier to train. These operations are briefly described in the following subsections.

Principal Component Analysis. Principal component analysis of complex system inputs and responses is aimed at decomposing the motions into their essential constituent parts. A special example of this is the modal decomposition of linear systems, and analogous decompositions can be defined for nonlinear systems using, for example, singular value decomposition (SVD), or a principal component analysis ANN.

SVD is described in detail, for example, in Golub and Van Loan (1983). It can be used to decompose linear or nonlinear structural motions in the following way. Let X be an $N \times n$ matrix representing the motion of a mechanical system at n transducer locations and at N consecutive times. The form of the SVD is

$$X = UWV^T \equiv uwv^T \qquad (5)$$

V and W describe characteristic shapes present in X and their corresponding amplitudes. The columns of U are filtered versions of the motions in X. The approximate equality on the right indicates that some components of the representation can be eliminated, and still maintain a good approximation to X. It is the evolution of the columns in u (a reduced form of U) which we seek to simulate.

Modal Filtering. Modal decomposition of complex system motions is often used for the simplification of mechanical system response when the model for the system is assumed linear or approximately linear. In fact though, the modal decomposition can be used on any collection of data; its purpose is to break the data into simple narrowband components, thereby simplifying system characterization, and perhaps simulation. When such a decomposition is used on nonlinear system or general system data, it is often referred to as modal filtering.

The form of a modal filter is similar to the SVD, but the means for obtaining the filter factors is much different. Meirovitch (1990) describes the operations included in the definition and use of a modal filter. The form of the modal filter can be expressed

$$X = US\phi^T \cong us\phi^T \tag{6}$$

where X is the same mechanical system motion representation as above, the columns of ϕ represent the characteristic shapes of the system, and the diagonal matrix S contains normalizing factors. When X comes from a linear system the columns of U are the linear modal components of the system motion. As with the SVD, the approximate equality on the right indicates that some components of the representation can be eliminated, and still maintain a good approximation to X, and it is the evolution of the columns in u (a reduced form of U) which we seek to simulate.

Elimination of Statistically Dependent Components. Under certain circumstances, some of the components produced during principal component analysis are perfectly statistically dependent upon others. For the sake of efficiency, we seek to eliminate statistically dependent components from the set to be modeled, then reintroduce these components during physical system simulation. In this way, ANN modeling of structural behavior is simplified.

When a dependency exists, it can be characterized using the conditional expected value of the variables in one column of u given values in another column of u. This requires approximation of a joint probability density function (pdf) of the data, and this can be obtained using the kernel density estimator. (See Silverman, 1986.)

The effect of eliminating components of motion that are completely dependent on other components is to eliminate some columns in the matrix $\boldsymbol{u}$. Denote the reduced matrix u_r; our objective is to model the evolution of the columns of u_r with an ANN.

Rosenblatt Transform. The previous step produces a description of the motion of a complex structure in terms of a set of components, none of which is completely statistically dependent on others. We can further transform the components, the columns of u_r, into signals that are statistically independent with Gaussian distributions. The transformation that accomplishes this is the Rosenblatt transform. (See Rosenblatt, 1952.) The complete form of the transform cannot be written here, but it can be denoted

$$z = T(u_r) \qquad u_r = T^{-1}(z) \tag{7}$$

The second expression indicates that the Rosenblatt transformation is uniquely invertible. The matrix z is the same size as the matrix u_r with the same number of nonzero columns. It is the evolution of the values in these columns that is to be simulated with ANNs. Because the columns in z are statistically independent, we need only to create ANN models for signals in individual columns.

Framework for Engineering Applications

Sandia National Laboratories performs analyses of the physical responses of complex systems for a variety of applications, including geothermal, solar, and wind energy systems, nuclear reactors, defense nuclear facilities, nuclear weapons, aerospace systems, telecommunication systems, transportation of hazardous materials, and storage of hazardous wastes. For these applications, Sandia has developed a variety of computer programs that perform various functions needed in design, testing, manufacturing, and safety or risk assessment.

Table I presents a summary of some of the principal phenomenological codes that have been developed at Sandia for various applications involving risk assessment, and provides an estimate for each code of the central processor unit (CPU) times required for execution on its usual platform. It is clear that for some of these codes, using today's computers, it would not be practical to run more than a few calculations for a given analysis because of the long running times. The complexities of the physical processes being modeled make it necessary to utilize fairly detailed, and therefore computer time-intensive, phenomenological models.

In the field of risk assessment, as in other applications of engineering analysis, recent studies have focused not only on evaluating system responses over a broad range of accident scenarios, but also on quantifying the uncertainties associated with our imprecise or incomplete knowledge of material properties and physical phenomena (Reactor Risk Reference Document, 1987). Standard Monte Carlo or Latin hypercube sampling processes used in risk assessments (Iman and Shortencarier, 1983) generate thousands, if not hundreds of thousands, of variations of parameters that are input to the phenomenological codes. Ideally, one would want to run the codes for each variation of parameters.

Since tomorrow's computers are projected to be much faster than today's, there is undoubtedly potential for a greater number of simulations to be performed in support of risk assessments and other applications such as design optimization. However, the tendency in the future will be to increase the level of detail of the models in order to improve accuracy, fidelity, and predictive capability rather than to maintain the current level of detail and enjoy the savings in computer time. Thus, these applications will continue to be subject to the limitations of computer processing times.

Given that we must work with a limited number of detailed code calculations, there is a strong need for a rational means for optimizing the choice of parameter values that are input to the code calculations. Optimization of parameter choices is required in order to obtain the desired characterization of the system with a minimum number of detailed computations. One way to accomplish this is to develop less detailed, faster-running models of the system that are sufficiently reliable to indicate where within the parameter space the most significant responses are likely to occur. In order for these less detailed, faster-running codes to provide adequate estimations of the responses, they must be calibrated to results from the more detailed codes. Artificial neural networks provide a convenient means for obtaining this calibration, and once obtained, for estimating the system responses in a very time-efficient manner.

Numerical Examples

Example 1. Let us consider the example problem of Figure 4, which is representative of some of the types of systems we consider at Sandia. The outer part of the system, or casing, is composed of a heat shield material and a metallic substrate. Inside the casing are several groups of components. One of these groups (the one shown in the figure) is composed of a cylindrical metal housing inside of which there are four small components that are partially encapsulated in a foam. The housing is attached by bolts to a baseplate which is fastened to the casing. Part of the assembly is exposed to a fire that is assumed to be uniform in temperature. The remainder is protected from the fire by being immersed in a semiconducting medium. The fire is assumed to endure until steady-state conditions are reached, after which it is presumed to be extinguished.

In this example, we explore the efficacy of a CNLS nonrecurrent neural network for estimating two attributes related to the thermal response of one of the components, namely Component 2 in the figure. The two attributes are: (1) the peak, or steady-state, temperature attained by the component, and (2) the time for the component to reach the halfway point between its initial and peak temperatures. These attributes, illustrated schematically in Figure 5, are representative of the much larger number of output variables that could be estimated by a neural network in order to more fully characterize the thermal responses of the components.

For this problem, we considered variations in three input parameters and investigated the ability of the network to predict the correct responses over the range of plausible variations. The input parameters were as follows: (1) fire temperature, (2) heat transfer coefficient for the immersing medium, and (3) exposure pattern. Fire temperatures were considered to be uniformly distributed between 550 and 2750 °C and heat transfer coefficients uniformly between 0.0 and 0.1 cal/cm^2-s-°C. Exposure patterns involved various orientations of the immersion plane with respect to the system. A feature of the ANN used in this application is that all inputs and outputs are normalized for network training and use.

We obtained 130 exemplars of "correct" behavior by modeling the system with a finite element mesh and running one of the thermal analysis codes listed in Table I for 130 different combinations of the input parameters. We selected the TEMPRA code [6] for this simulation because it is capable of providing relatively accurate thermal responses without very large expenditures of computer time.

One of the issues to be explored is the number of exemplars required in order to provide adequate simulation of the responses. To obtain a benchmark against which to measure other results, we first investigated the accuracy of the network when all 130 TEMPRA runs were used both as exemplars and as test cases. The results are shown in Figures 6 and 7. To obtain these figures, we iterated upon the number of radial basis functions, the locations of the centers, and the widths until best results were obtained. The results shown correspond to 8 basis functions.

Whereas the results for peak temperature in Figure 6 indicate close agreement between TEMPRA and the neural network, the results for the time to mid-temperature in Figure 7 illustrate an important aspect of the behavior of neural networks that can lead to undesirable results if proper accounting is not taken. When the responses tend to occur predominantly at one end of the response scale, the results tend to be skewed in that direction. Important, but less numerous, results at the other end of the scale can be inadvertently underemphasized. This tendency seems to be a product of the least squares algorithm that is used to minimize differences between the network response and that of the simulation code.

One way to avoid this bias is to redefine the output parameter so that the response is more evenly distributed. In the case of the example problem, the situation is corrected by redefining the time to mid-temperature so as to include a normalizing factor that contains the fire temperature, as follows:

$$t^{*}_{0.5} = t_{0.5} \bullet \left(\frac{T^4_{fire} - T^4_{in}}{T^4_{in}} \right) \quad (8)$$

This normalization recognizes the fact that to a first level of approximation, the time for a component to achieve a given level of temperature is inversely proportional to the radiative heat flux from the fire to the system, and the latter is proportional to the fourth power of the fire temperature. The normalization improves the distribution of exemplars. Figure 8 illustrates that the results from the neural network are greatly improved when the output variable is redefined in this way.

To explore whether reasonable results can be obtained with fewer exemplars, we reperformed the analysis using only a fraction of the original 130 TEMPRA runs as training exemplars and the remainder of the runs as test cases. Figures 9 and 10 show the results obtained by using only 20 exemplars and 110 test cases. The exemplars were taken to be those TEMPRA runs for which the input parameters were at the extremes of their ranges. Thus, they represent various combinations of fire temperatures taken at either 550 or 2750 °C, heat transfer coefficients at either 0.0 or 0.1 cal/cm^2-s-°C, and exposure patterns involving either minimal or maximal coverage of the surface. The results indicate that the estimating capability of the neural network remains fairly high even with such a very small number of training exemplars. This result can be crucially important for engineering analyses when it is difficult to obtain a large number of simulations to be used as exemplars because of computer time.

Further evidence of the potential of neural networks is provided by comparing Figures 9 and 10 with Figures 11 and 12. We obtained the latter two figures by using a response surface approach to estimate the peak temperature and normalized time to mid-temperature for the component under investigation. The response surface for each output variable consisted of a truncated Taylor series containing the first-order derivatives of the output with respect to each of the inputs and the various first-order cross derivatives. Each Taylor series contained 20 terms, and the corresponding coefficients were obtained by fitting the series to the results of the 20 TEMPRA runs that were used as training exemplars for the neural network. Put in other terms, the points labeled as "estimates" and "computations" in Figures 11 and 12 correspond to the points labeled as "test cases" and "exemplars", respectively, in Figures 9 and 10. For this case, the results from the neural network appear to be superior to the results from the response surface.

Example 2. We simulate in this example the motion of a simple 10 degree-of-freedom system excited with a Gaussian white noise. Figure 13 shows the system. The recurrent CNLS net was used in the simulation, and training data were generated by computing response over 6000 time steps; excitation and responses at ten locations were recorded. Figure 14 shows the response at one of the degrees of freedom. The responses were placed in a matrix X as referred to in the previous section on data reduction, and its SVD was computed. The singular values of the response indicate that accuracy of about 89% should be achieved by simulating the system response with its first three components. All the principal components of the response are narrowband random processes. None of the three components is statistically dependent, so none was removed. The system is linear, so its response is Gaussian, therefore, the Rosenblatt transform was not used on the three components in the simulation. (Linearity is of no special benefit in the CNLS net modeling process since the framework for the CNLS net is nonlinear.) The first three components of the response were modeled with recurrent CNLS nets. Each recurrent CNLS net had four inputs: external excitation at time index j, component response at time index j, component response one-quarter cycle in the past (Recall that all the components are narrowband.), and component response one-half cycle in the past. The recurrent CNLS net output was the component response at time index j+1.

The entire system was tested using data generated over 1000 steps of response computation. The initial conditions and excitation were used to start then execute a response simulation with CNLS nets in the space of $\boldsymbol{u}$. The first column of $\boldsymbol{u}$ from the test data and the first column from the simulated data are compared in Figure 15. This is the dominant component of the response. The match is good, particularly in view of the fact that the simulation is iterated. The simulated response is now reconstructed by substituting into Eq. (5) using the simulated $\boldsymbol{u}$, and the results are compared to the test response at one of the degrees of freedom. This is shown in Figure 16. Of course, the match is quite good since the dominant component is well simulated.

Conclusions

We have described in this paper some frameworks for the simulation of complex physical systems with artificial neural networks, in particular, the connectionist normalized linear spline network. Both nonrecurrent and recurrent applications have been described. Numerical examples were presented that show the accuracy of the approximations we achieved. For our purposes this level of accuracy is adequate, though greater accuracy could be achieved by increasing the size of the ANNs, increasing the number of exemplars, and increasing the training times. Though we have not simulated the recurrent behavior of extremely complex systems, we believe that there is very good potential to do so accurately because complex system response can be separated into simple (though perhaps nonlinear) components, and we have been successful at simulating such components.

Among many other uses, the application of ANNs to the simulation of complex systems is especially important in probabilistic risk assessment where numerous realizations of system response must be generated during the execution of a Monte Carlo analysis. In particular, the use of recurrent ANNs is important in the approximate assessment of system response where events of significance occur at arbitrary times. With continued development, ANNs will prove to be especially useful for the accurate and efficient generation of high fidelity simulations. Current experience indicates that satisfactory simulation of complex systems can be achieved with ANNs with computation times up to two orders of magnitude lower than phenomenological models.

References

1. Benjamin, A. S., Beraun, R., Brown, N. N., and Sherman, M. P., (1995), Evaluation of Conductive, Radiative, Chemical, and Convective Heat Transfer in Complex Systems Using a Fast-Running, Implicit, Lumped Capacitance Formulation. National Heat Transfer Conference. American Society of Mechanical Engineers. Portland, Oregon.
2. Benjamin, A. S., and Altman, B. S., (1995), Evaluation of Nonlinear Structural Responses Using a Fast-Running Spring-Mass Formulation. International Conference on Computational Engineering Science. Mauna Lani, Hawaii.
3. Freeman, J. A., Skapura, D. M., (1991), *Neural Networks, Algorithms, Applications, and Programming Techniques*, Addison-Wesley Publishing Company, Reading Massachusetts.

4. Gartling, D. K., and Hogan, R. E, (1994), "COYOTE 2 - A Finite Element Computer Program for Nonlinear Heat Conduction Problems," SAND94-1173, Sandia National Laboratories.
5. Golub, G. H., Van Loan, C. F., *Matrix Computations*, The Johns Hopkins University Press, Baltimore, Maryland, 1983.
6. Helton, J. C., Anderson, D. R., Baker, B. L., et al., (1995), Effect of Alternative Conceptual Models in a Preliminary Performance Assessment for the Waste Isolation Pilot Plant. Nuclear Engineering and Design; 154:251-344.
7. Iman, R. L., and Shortencarier, M. J., (1983), "A FORTRAN 77 Program and Users Guide for the Generation of Latin Hypercube and Random Samples for Use with Computer Models," NUREG/CR-3624, SAND83-2365, Sandia National Laboratories.
8. Jones, R. D., et. al., (1990), "Nonlinear Adaptive Networks: A Little Theory, A Few Applications," *Cognitive Modeling in System Control*, The Santa Fe Institute.
9. Meirovitch, L., (1990), *Dynamics and Control of Structures*, Wiley, New York, pp. 232-239, pp. 255-267.
10. Moody, J., Darken, C. J., (1989), "Fast Learning in Networks of Locally Tuned Processing Units," *Neural Computation*, 1, 281-294.
11. Paez, T., (1993), "Neural Networks in Mechanical System Simulation, Identification, and Assessment," *Shock and Vibration*, 5, n. 2, 177-199.
12. Paez, T. L., (1994), "Simulation of Structural Response Using a Recurrent Radial Basis Function Network," *Proceedings of the First World Conference on Structural Control*, Pasadena, California.
13. Paez, T. L., (1995), "Simulation of Large Systems with Neural Networks," *Proceedings of the 65th Shock and Vibration Symposium*, V. I, SAVIAC, San Diego.
14. Paez, T. L., (1996), "Simulation of Nonlinear Structures with Artificial Neural Networks," *Proceedings of the 11th Engineering Mechanics Conference*, ASCE, Fort Lauderdale.
15. Reactor Risk Reference Document, (1987), NUREG-1150, U.S. Nuclear Regulatory Commission.
16. Roach, P. J., (1993), The SECO Suite of Codes for Site Performance Assessment. Proceedings of 4th International Conference on High Level Radioactive Waste Management, American Nuclear Society.
17. Summers, R. M., et al., (1995), MELCOR Computer Code Manuals, Vol. 1, Primer and Users Guides, Version 1.8.3. NUREG/CR-6119, SAND93-2185, Sandia National Laboratories.
18. Taylor, L. M., and Flanagan, D. P., (1989), "PRONTO 3D, A Three-Dimensional Transient Solid Dynamics Program," SAND87-1912, Sandia National Laboratories.
19. Vanluchene, R. D., Sun, R., (1990), "Neural Networks in Structural Engineering," *Microcomputers in Civil Engineering*, 5, n. 3, 207-215.
20. Widrow, B., Stearns, S. D., (1985), *Adaptive Signal Processing*, Prentice-Hall.

Acknowledgment

This work was sponsored by the Department of Energy under contract No. DE-AC04-94AL85000.

Table I. Comparison of Some of Sandia's Phenomenological Computer Codes Used to Support Risk Assessments

Code Name	Ref.	Risk Application	Function	Typical CPU*
MELCOR	[16]	Nuclear Reactors	Progression of core degradation and containment response for severe accidents	5-10 Hours/ Intel Pentium PC
BRAGFLO	[6]	Radioactive Waste Storage	Multiphase flow of gas and brine through a porous reservoir	3-6 Hours/ Digital VAX-α
SECO-TRANSPORT	[15]	Radioactive Waste Storage	Fluid flow and transport of radionuclides in fractured porous media	3-6 Hours Digital VAX-α
TEMPRA	[1]	Nuclear Weapons	System thermal responses to timewise and spatially dependent fires	20-40 Minutes Sun SPARC-20
STRESS	[2]	Nuclear Weapons	System structural responses to impacts and punctures	40-80 Minutes Sun SPARC-20
COYOTE	[4]	Variety of Applications	Finite element analysis of system thermal responses	10-20 Hours IBM RS-6000
PRONTO	[17]	Variety of Applications	Finite element analysis of system dynamic structural responses	20-40 Hours Cray-YMP

* Per accident scenario

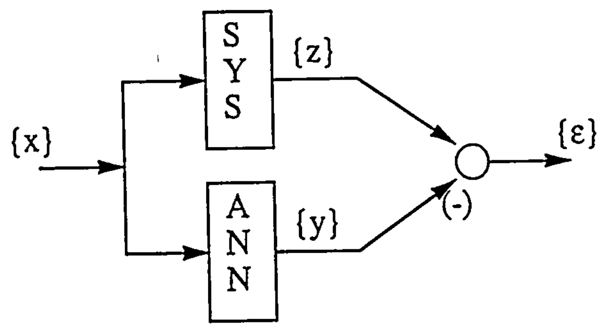

Figure 1. Block diagram of a system and an artificial neural network.

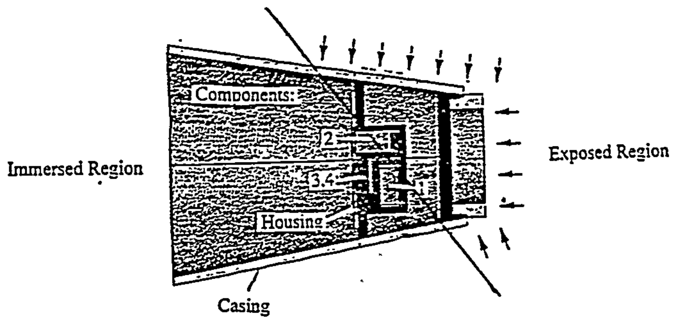

Figure 4. Test Problem Configuration.

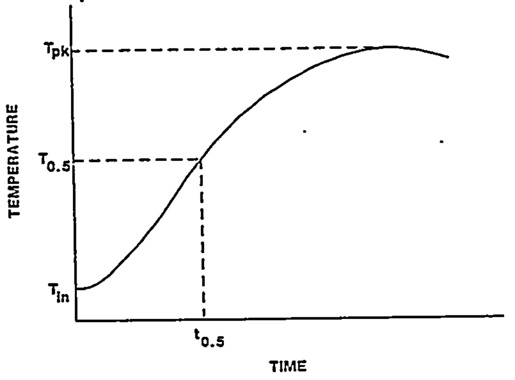

Figure 5. Schematic of Thermal Response Variables Used for Training and Testing of Neural Network.

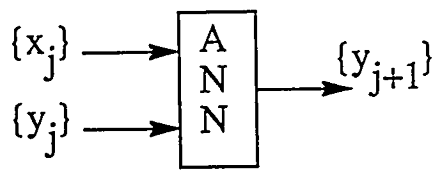

Figure 2. Block diagram of a recurrent ANN.

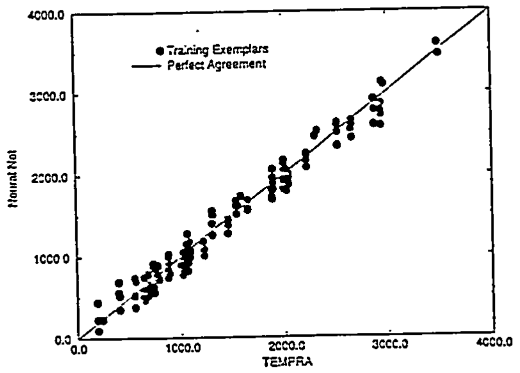

Figure 6. Comparison of Neural Network and TEMPRA Code Results for Peak Temperature: 130 Training Exemplars. (Temperatures in °F)

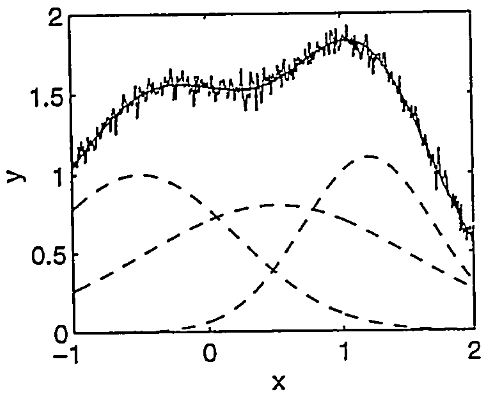

Figure 3. Motivational example for the CNLS network.

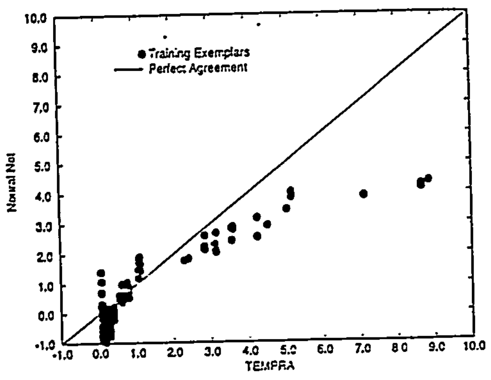

Figure 7. Comparison of Neural Network and TEMPRA Code Results for Time to Mid-Temperature: 130 Training Exemplars. (Time in Hr)

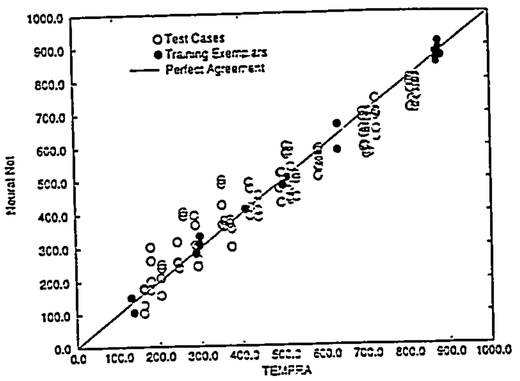

Figure 10. Comparison of Neural Network and TEMPRA Code Results for Normalized Time to Mid-Temperature: 20 Training Exemplars, 110 Test Cases. (Time in Hr)

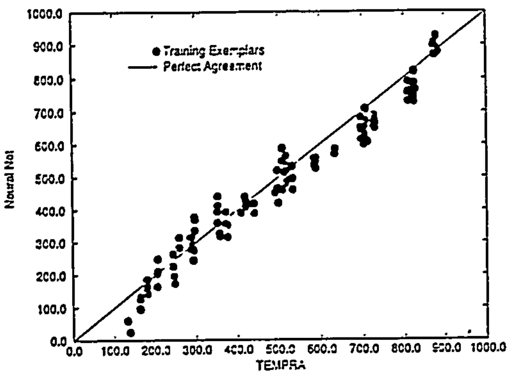

Figure 8. Comparison of Neural Network and TEMPRA Code Results for Normalized Time to Mid-Temperature: 130 Training Exemplars. (Time in Hr)

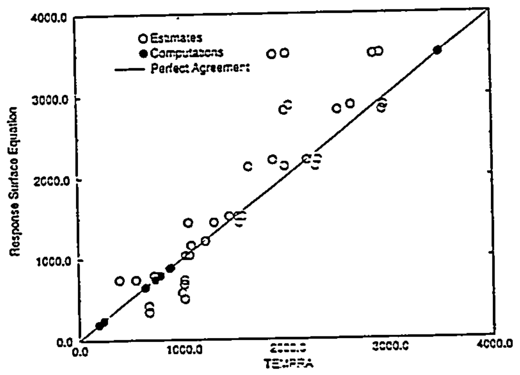

Figure 11. Comparison of Response Surface and TEMPRA Code Results for Peak Temperature: 20 Computations, 110 Estimates. (Temperatures in °F)

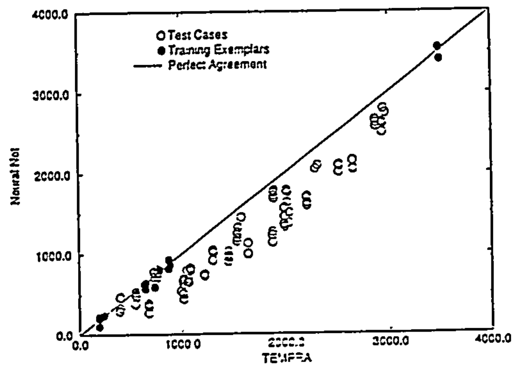

Figure 9. Comparison of Neural Network and TEMPRA Code Results for Peak Temperature: 20 Training Exemplars, 110 Test Cases. (Temperatures in °F)

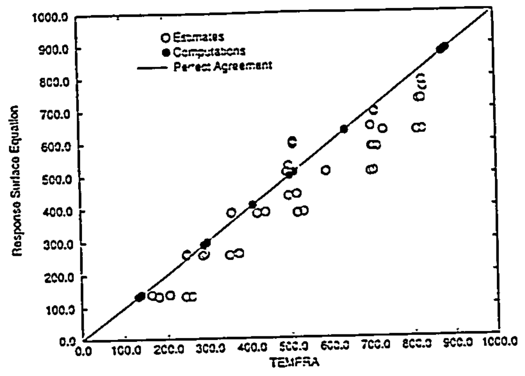

Figure 12. Comparison of Response Surface and TEMPRA Code Results for Normalized Time to Mid-Temperature: 20 Computations, 110 Estimates. (Time in Hr)

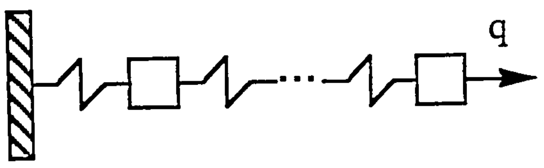

Figure 13. A simple system.

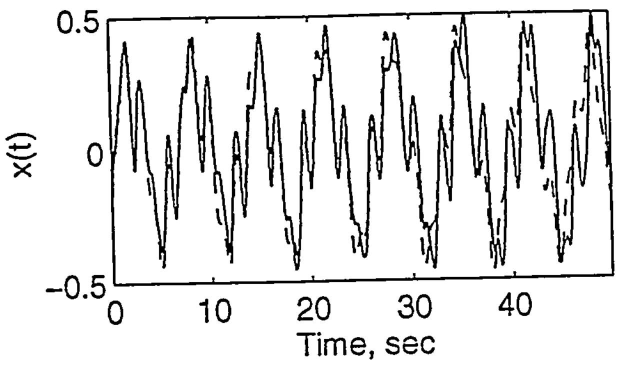

Figure 16. Test and simulated responses at a point in the simple system.

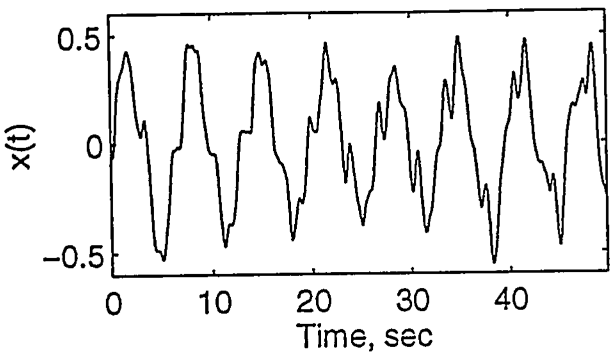

Figure 14. Response to white noise input.

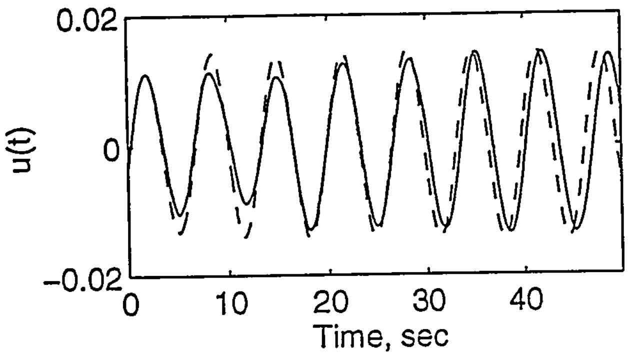

Figure 15. Comparison of test and CNLS simulated responses - Component 1.