AIAA-98-1780

DESIGN OF NEURAL NETWORKS FOR FAST CONVERGENCE AND ACCURACY

Peiman G. Maghami* and Dean W. Sparks, Jr.†

NASA Langley Research Center, Hampton, VA 23681-0001

Abstract

A novel procedure for the design and training of artificial neural networks, used for rapid and efficient controls and dynamics design and analysis for flexible space systems, has been developed. Artificial neural networks are employed to provide a means of evaluating the impact of design changes rapidly. Specifically, two-layer feedforward neural networks are designed to approximate the functional relationship between the component/spacecraft design changes and measures of its performance. A training algorithm, based on statistical sampling theory, is presented, which guarantees that the trained networks provide a designer-specified degree of accuracy in mapping the functional relationship. Within each iteration of this statistical-based algorithm, a sequential design algorithm is used for the design and training of the feedforward network to provide rapid convergence to the network goals. Here, at each sequence a new network is trained to minimize the error of previous network. The design algorithm attempts to avoid the local minima phenomenon that hampers the traditional network training. A numerical example is performed on a spacecraft application in order to demonstrate the feasibility of the proposed approach.

Introduction

The overall design process for aerospace systems typically consists of the following steps: design, analysis, and evaluation. If the evaluation is not satisfactory, the process is repeated until a satisfactory design is obtained. Dynamics and controls analyses, which define the critical performance of any aerospace system are particularly important. Generally, all aerospace systems experience excitations resulting from internal and operational disturbances, such as instrument scanning in space systems or aerodynamic turbulence in aircraft. These excitations can potentially interfere with the mission of the system. For example, in space systems, excessive vibrations could be detrimental to its science instruments which usually require consistently steady pointing in a specified direction for a prescribed time duration, or excessive vibrations due to turbulent aerodynamics could diminish the ride quality or safety of an aircraft. Typically, in the course of the design of a spacecraft, as the definitions and the designs of the spacecraft and its components mature, several detailed dynamics and controls analyses are performed in order to insure that all mission requirements are being met. These analyses, although necessary, have historically been very time consuming and costly due to the large size of the aerospace system analysis model, large number of disturbance scenarios involved, and the extent of time domain simulations that need to be carried out. For example, a typical pointing performance analysis for a space system might require several months or more, which can amount to a considerable drain on the time and resources of a space mission.

It is anticipated that artificial neural networks can be used to significantly speed up the design and analysis process of aerospace systems. However, there are certain drawbacks associated with neural networks. These include, the time-consuming nature of the training process, training difficulties, such as optimization problems, and a lack of a meaningful way to establish network accuracy. The focus of this paper is to address these specific issues, and develop a methodology for efficient and fast training neural networks, with specified accuracy, to approximate the functional relationships between design change parameters (be they structural or material properties, in the disturbance environment, or in the control system design) and the performance of the system/component. A critical concern with any

* Senior Research Engineer, Guidance and Control Branch, Senior Member, AIAA.

† Aerospace Technologist, Guidance and Control Branch.

approximation is its accuracy. Typical neural network training involves the use of a select set of input and output data, taken from the functional relationship to be approximated. If these training points are chosen judiciously, the trained neural network should give a very good approximation. However, there is no guarantee that the neural network will continue to give a good approximation of the relationship for those points not in the training set. The design methodology presented in this paper addresses this problem, in terms of allowing the design of neural networks to a specific level of accuracy (in terms of approximating relationships), for a given statistical confidence level, and accounting for input and output data not used in the original training set. Moreover, a sequential training algorithm is presented for a two-layer feedforward network, which should provide benefits, such as enhanced training speed, and automated network architecture design. Numerical examples based on a spacecraft application are carried out to demonstrate the feasibility of the proposed neural network design methodology.

The paper is organized as follows. Following this introduction section, brief descriptions on typical (conventional) dynamics analysis will be given. Next, discussions on neural networks, their use to approximate functional relationships, and a typical design procedure, will be presented. Then, the new neural network methodology, given in a step by step format, will be presented, followed by numerical examples of the proposed approach applied to a NASA spacecraft model. Finally, a concluding remarks section will close the paper.

Typical Dynamics and Controls Analysis

Whatever type of analysis to be performed, it would be highly beneficial to the analyst to be able to rapidly assess the effects on overall system performance due to the almost inevitable design changes that a system will undergo during its lifetime. During design phase of the spacecraft almost all components go through changes, with each change having the potential to affect the performance of the spacecraft to some degree. In many instances, these changes are expected to affect the performance of the spacecraft so much as to warrant a partial or full analysis of its performance. In the area of spacecraft dynamics and controls, these type of changes include, changes in the inertia or flexibility of the structural components which would affect the dynamic characteristics of the spacecraft; changes in the control system design, hardware, and software; or changes in the characteristics of the external and internal disturbances that may act on the spacecraft in orbit. Now, depending on the nature and extent of these changes, there may be a need to reevaluate the controlled dynamical performance of the system. The dynamical performance could be in terms of pointing and jitter performance, tracking performance, closed-loop stability margin, and many other forms. The computational time and cost associated with each of these performance analysis may be substantial. For example, to evaluate the effects of changing the location of an instrument within the spacecraft bus on the pointing stability of another instrument, a full finite element analysis, followed by a possible controller redesign, closed-loop simulation of spacecraft response for possibly all disturbance scenarios, and a jitter and stability analysis, are required. Perhaps, even more analysis would be required if the controller has to be redesigned. The cost of such analyses can be exorbitant, particularly, when they have to be repeated several times during the design phase. One approach to this problem is to use artificial neural networks (ANNs), particularly feedforward networks, for rapid system analysis and design.

Rapid Analysis and Design with ANN

The motivation behind the use of an ANN is to speed up the analysis or design process substantially. The main use of an ANN is in its ability to approximate functional relationships, particularly nonlinear relationships. This can be a static relationship, one that does not involve time explicitly, or a dynamics relationship, which explicitly involves time. For an ANN there is no distinction between a static or dynamic map, there is just input/output training data. For example, an ANN would be designed to approximate the mapping between the first instrument location and the pointing performance of the second instrument. Once such a network is trained, the pointing performance of the second instrument for a specified location of the first instrument may be easily and almost instantaneously obtained by simulating the ANN for the one input point, corresponding to the location of the first instrument. It is this advantage of ANNs that promises savings in the overall design and analysis time. Although the initial training time for an ANN may be long, it can be performed during off hours, in a semi-automated manner, without much involvement by the designer(s). Another example could be an ANN that would be designed to approximate the dynamic behavior of a nonlinear component, e.g., nonlinear reaction wheel with friction. Dynamic approximations via ANN is achieved by using time delays and feedback of the output back to the input, which is defined as recurrence. Such networks are referred to as recurrent networks [1-2].

The successful design of an ANN depends on the proper training of the network as well as the ability to characterize the accuracy of its approximation. The

training of the network involves the judicious selection of points in the input variable space, which along with the corresponding output points, constitute the training set. For example, the design and training of an ANN for mapping a component parameter to spacecraft performance relationship requires a number of design points for use in training. These points generally span the range of the changes that the component parameter is allowed to have, many of which would cause significant changes in the system model. The proper training of the network is the key to its ability to provide a good mapping. However, this is possible only if one can quantify the degree of accuracy of the approximation of the ANN, particularly for points in the parameter space not included in the training set.

Artificial Neural Networks (ANNs)

Artificial neural networks (ANNs) have grown into a large field since their inception, and a complete discussion on them is beyond the scope of this paper. Instead, this section will present a very brief description on ANNs. ANNs were developed to mimic some of the processes of the human brain. They consist of groups of elements (called neurons) which perform specific computations on incoming data, with interconnections which permit data flow from one group of neurons to the next, similar to the way groups of real neurons receive and transmit information through dendrites and axons, respectively, in the brain. Like their biological counterparts, ANNs can be trained to perform a variety of tasks, such as modeling functional relationships. The ANN, when presented with appropriate input and output data related to a specific functional relationship, can adjust itself such that it can give a good representation of that relationship. This feature is particularly useful when the relationship is nonlinear and/or not well defined, and thus difficult to model by conventional means. Also ANNs, by their very nature, are a perfect fit for efficient parallel computations on digital computers. Though there are several types of ANNs, in this paper, only the feedforward ANN will be discussed.

A typical feedforward ANN is depicted in figure 1, with m inputs and n_p outputs, and each circle representing a single neuron. The name feedforward implies that

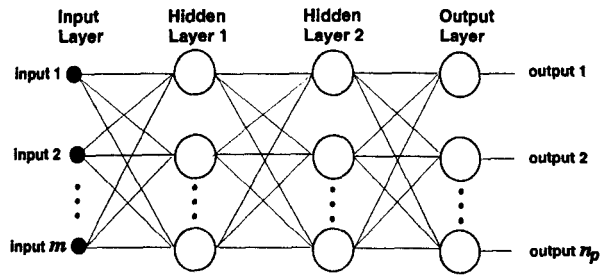

Figure 1. Typical Feedforward ANN.

the data flow is one way and there are no feedback paths between neurons. The output of each neuron from one column is an input to each neuron of the next column. Using the typical naming convention, each column of neurons is called a layer, the initial column where the inputs come into the ANN is called the input layer, and the last layer, i.e., where the outputs come out of the ANN, is denoted as the output layer. All other layers in between are called hidden layers. These ANNs can have as many layers as desired, and each hidden layer can have as many neurons as desired. Each neuron can be modeled as shown in figure 2, with n being the number of inputs to the neuron. Associated with

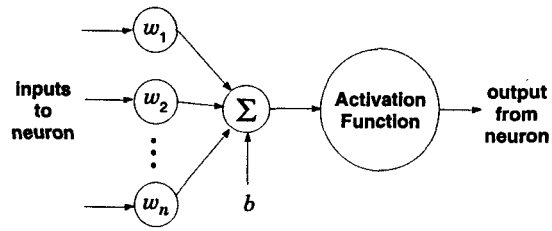

Figure 2. Representation of a neuron in the feedforward ANN.

each of the n inputs is some adjustable scalar weight, w_i, $i = 1, 2, ..., n$, which multiplies that input. In addition, an adjustable bias value, b, can be added to the summed scaled inputs. These combined inputs are then fed into an activation function, which produces the output of the neuron. The activation function can take on many forms to shape the output; three of the more common functions are linear, log sigmoid and binary, as shown in figure 3. The linear activation function simply outputs the input; the log sigmoid function is a nonlinear function of the input, with output values between 0 and 1; while the binary activation function outputs a +1 for non-negative inputs and a -1 for nega-

tive inputs. During training, the set of weights and bias

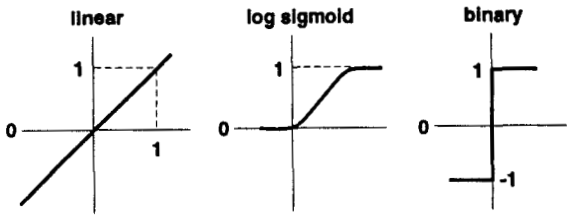

Figure 3. Three common activation functions.

terms associated with the neurons are adjusted until the output of the ANN matches, to within some specified level of tolerance, the true outputs of the function, for the same inputs.

Training of a Feedforward ANN

The objective is to design and train a feedforward network to map the relationship between an $n_c \times 1$ input vector, δ, and an $n_p \times 1$ output vector y_p. In this paper, a feedforward network, like the one shown in figure 1, but with only one hidden layer, is considered. It has been shown in the literature that a feedforward network, with only one hidden layer, can approximate a continuous function to any degree of accuracy[1–3]. It is obvious that this capability carries over to networks with more than one hidden layer. Furthermore, assume that the activation function for the output layer is a pure linear. The output layer has n_p nodes, corresponding to the elements of vector y_p. The number of nodes in the hidden layer, n_h is arbitrary, however, it has to be large enough to guarantee convergence of the network to the functional relationship that it is designed to approximate, but not too large as to cause overmapping. The network equation for this two-layer feedforward network is given by

$$Y_n = W_2(f(W_1\Delta + b_1)) + b_2 \qquad (1)$$

Here, W_1 and W_2 represent the $n_h \times n_c$ and $n_p \times n_h$ weighting matrices for the hidden and output layers, respectively; b_1 and b_2 are respectively the bias vectors for the two layers; f represents the activation function for the hidden layer; Δ is an $n_c \times q$ matrix denoting the collection of q $n_c \times 1$ input vectors; and Y_n is an $n_p \times q$ matrix representing the output of the network. Once the number of nodes in the hidden layer has been chosen, the network design is reduced to adjusting the weighting coefficient matrices, W_1 and W_2, and the weighting bias vectors, b_1 and b_2. The parameters of feedforward networks are usually adjusted using a gradient method, named the back propagation method[4,5], or a pseudo-Newtonian approach, such as the Levenberg-Marquardt technique [6]. Typically, in these methods, the weighting matrices and bias vectors are adjusted to minimize some cost function, which is a function of the error of the network. The network error is defined as the difference between the output of the true system and that of its neural network approximation, for a given set of inputs. The cost function is usually taken as the sum squared error of the network over all of the input sample points. If q sets of points are used for training the network, then the cost function, in terms of the sum squared error (SSE) of the network, can be written as

$$E = \sum_{k=1}^{qn_p} e(k)^2 = \sum_{r=1}^{q}\sum_{j=1}^{n_p} (Y_d(j,r) - Y_n(j,r))^2 \qquad (2)$$

where Y_d is a n_p x q matrix of the desired outputs. The typical procedure is to keep updating the weights and biases until the error E goes below some specified tolerance level. At this point, the feedforward network is considered trained.

The use of feedforward ANNs has some advantages over the conventional approximation techniques, such as polynomials and splines. For example, polynomials are hard to implement in hardware due to signal saturation, and if they are of higher order, there may be stability problems in determining the coefficients. ANNs, on the other hand, are very amenable to hardware implementation. As a matter of fact, to date, several VLSI chips based on multilayer neural network architecture are available[7–8]. Also, because of the highly interconnected and coupled nature of ANNs, they are rather robust to hardware failures, since the weight is distributed among many nodes.

There are a few problems associated with neural networks. One has to do with the rate of convergence during training, in other words, training time. The second problem is the proper architecture, e.g., node numbers and layers, to use. Third, is the tendency of the steepest descent technique, and even, pseudo-Newton methods, which are used in the training process, to get stuck in local minima. This causes training problems and contribute substantially to the time it takes to train a network. Finally, the training of the network is based solely on the given input and output data points; there is no guarantee, when the network is given new input data points, that the resulting output data will approximate the corresponding true output data to within the specified error tolerance to which the network was designed. In the next section, a new and novel design procedure is presented to address these problems with neural networks.

Network Training for Accuracy and Fast Convergence

Mathematical Modeling

As mentioned earlier, ANNs may be used to approximate the functional relationship between changes in the system components and the spacecraft performance measures. In the area of dynamics and control, these changes may include those effecting the structural model of the system, the control system model, or the disturbance models. A typical block diagram of a controlled spacecraft is given in figure 4. Here, the elements of the vector δ_s could represent a wide variety of potential changes in the structural model, such as variation in size for structural element, frequency uncertainty/variation in flexible modes, and many others changes. Those changes which could impact the structural model must necessarily involve some redistribution of mass, flexibility, or damping. The elements of δ_d could represent the changes in the magnitude and phase characteristics as well as the location (on the structure) of the external and internal disturbances that are to be considered. The external disturbances, such as gravity gradient torques, atmospheric drag torques, and etc., are fairly well understood. However, they generally depend on the geometry and inertia of the spacecraft which may undergo several changes during the design phase. The internal disturbances are typically the result of scanning or spinning instruments or components whose characteristics may change radically in the design phase. The

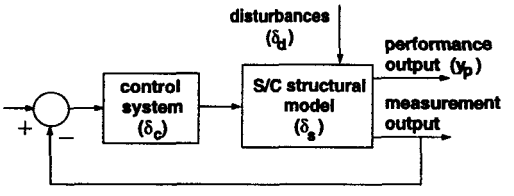

Figure 4. Typical block diagram of controlled spacecraft.

elements of δ_c could represent the changes in the control system model. Similar to the structural and disturbance models, the control system model may change several times during the design phase due to changes in the control system hardware characteristics, such as reaction wheels, rate gyros, star trackers, and etc., or changes in the system software, such as controller gains, dynamics (frequency content), saturation limits, and others.

Now, let δ be an $n_c \times 1$ vector representing the overall potential change vector, defined as

$$\delta = \begin{Bmatrix} \delta_s \\ \delta_d \\ \delta_c \end{Bmatrix} \tag{3}$$

The performance of the spacecraft, in the context of dynamics and control, may be divided into time-domain measures and frequency-domain measures. Time-domain performance measures are typically the pointing performance, jitter/stability performance, or tracking performance of the spacecraft or its instruments. These performance measures are usually obtained from time simulations of the controlled spacecraft model, represented by the feedback connection of the dynamic model of the spacecraft and the full control system model in the presence of external or internal disturbances. The frequency-domain performance measures are typically the stability margins, bode plots, closed-loop bandwidth, and other frequency-based stability and performance measures. The frequency-domain performance measures are generally obtained from the analysis of the linear closed-loop model of the spacecraft. Let, y_{pm} be an $r \times 1$ vector representing all time-domain and frequency-domain performance measures that are to considered for potential impact by design changes (a subset of vector y_p). Now, the problem is to design an ANN which can map, to a specified degree of accuracy, the relationship between the potential design change vector δ and the spacecraft performance measure vector y_{pm}.

Training with Fast Convergence

Here, a novel algorithm is presented to deal with the three problems associated with the training of ANNs, namely, size of the network, excessive training time, getting stuck in a local minima. The algorithm is a sequential algorithm, wherein at each step a new feedforward network is designed and trained which minimizes the current error function, which represents the level of achievement by all the previous ANNs. Assume that a ANN map is desired for a training data $[X, Y]$, where X represents an input sequence, and Y denotes an output sequence. The sequential training algorithm is summarized as follows:

a. Choose a feedforward network with one hidden layer. The hidden layer can be any type of layer, such as tan sigmoid, log sigmoid, etc. The output layer should be a pure linear layer.
b. Use a small to moderate number of nodes, n_0, for the hidden layer.
c. Based on a desired network error tolerance, e_{tol}, begin adjusting the parameters of the two layer feedforward network, namely, W_1^1, b_1^1, W_2^1, and b_2^1. Here, the subscript indicates the layer number and the superscript denotes the network number. Note that techniques, such as back-propagation procedure, a pseudo-Newtonian technique, such as the Levenberg-Marquardt algorithm, or any other optimization technique may be used to perform the

process of adjusting the network parameters to minimize the error.

d. Stop the training process as soon as the network error goal is achieved or some measure of the rate of decrease of the network error (learning rate), for example the decrease in the network error from some previous epoch to the current epoch, gets below a designer–defined level. If the network error goal is attained, the training process is finished, otherwise, continue with the algorithm.

e. Update the overall weighting and bias matrices, and activation functions

$$\begin{aligned} W_1^F &= W_1^1 \; ; \; B_1^F = B_1^1 \\ W_2^F &= W_2^1 \; ; \; B_2^F = B_2^1 \\ f^F &= f^1 \end{aligned} \tag{4}$$

f. Compute the current network error at each training point from Eq. (1) as follows:

$$D_1 = Y - \left[W_2^1\left(f^1\left(W_1^1 X + b_1^1\right)\right) + b_2^1\right] \tag{5}$$

where $f^1()$ represents the activation function used.

g. Use the pair $[X, D_1]$ as the new training data.

h. Choose a new feedforward network with the same architecture as the previous, i.e. one hidden layer and one output layer. The hidden layer can be any type of layer, such as tan sigmoid, log sigmoid, etc., and does not have to be of the same type as that for the previous network. The output layer should be a pure linear layer.

i. Use a small to moderate number of nodes, n_1, for the hidden layer. It is recommended to choose the node number randomly from a range defined by the designer. At any rate, it is best that the number nodes chosen is different from that used in the previous network. This would make sure that the number of design variables (degrees of freedom) changes from the previous problem, so that there is a lesser chance of continuously getting stuck in a local minima.

j. Based on the same desired network error tolerance, e_{tol}, begin adjusting the parameters of the two layer feedforward network, namely, W_1^2, b_1^2, W_2^2, and b_2^2.

k. If the network error goal is achieved, then form the overall network parameters as

$$\begin{aligned} W_1^F &= \begin{bmatrix} W_1^F \\ W_1^2 \end{bmatrix} \; ; \; B_1^F = \begin{bmatrix} B_1^F \\ B_1^2 \end{bmatrix} \\ W_2^F &= \begin{bmatrix} W_2^F & W_2^2 \end{bmatrix} \; ; \; B_2^F = B_2^F + B_2^2 \\ f^F &= \begin{bmatrix} f^F \\ f^2 \end{bmatrix} \end{aligned} \tag{6}$$

and the training process is finished.

l. If the rate of decrease of the network, as discussed in step (d), gets below a designer–defined level, stop the training process and update the current network parameters as

$$\begin{aligned} W_1^F &\leftarrow \begin{bmatrix} W_1^F \\ W_1^2 \end{bmatrix} \; ; \; B_1^F \leftarrow \begin{bmatrix} B_1^F \\ B_1^2 \end{bmatrix} \\ W_2^F &\leftarrow \begin{bmatrix} W_2^F & W_2^2 \end{bmatrix} \; ; \; B_2^F \leftarrow B_2^F + B_2^2 \\ f^F &\leftarrow \begin{bmatrix} f^F \\ f^2 \end{bmatrix} \end{aligned} \tag{7}$$

m. Compute the current network error at each training point as follows:

$$D_2 = D_1 - \left[W_2^2\left(f^2\left(W_1^2 X + b_1^2\right)\right) + b_2^2\right] \tag{8}$$

n. In the first variation of the algorithm, use the pair $[X, D_2]$ as the new training data, and repeat the algorithm, beginning from step (h), until one of three things happens: (i) network error goals are attained at some sequence; (ii) a designer–defined limit on the total number of training epochs (steps) is reached; or (iii) a designer-defined limit on the number of training sequences is reached.

o. In the second variation of the algorithm, repeat the steps, beginning from step (a), but using the overall weighting and bias matrices, and activation functions for the initial guess, i.e.,

$$\begin{aligned} W_1^1 &\leftarrow W_1^F \; ; \; B_1^1 \leftarrow B_1^F \\ W_2^1 &\leftarrow W_2^F \; ; \; B_2^1 \leftarrow B_2^F \\ f^1 &\leftarrow f^F \end{aligned} \tag{9}$$

This variation could lead to smaller overall networks (which satisfy the SSE goal), however, it can require substantially longer training time since the size of the network increases much faster as compared to the first variation of the algorithm.

As mentioned previously, even if the ANN is trained such that the network error is exactly zero there are no guarantees that it can provide an accurate approximation for points not in the training set. Of course, as the number of training points increase one expects that the accuracy of the network would improve. However, there is no systematic way of establishing a priori this increase in the accuracy of the network or ascertaining that it does occur. Nonetheless, for an ANN to be useful in approximating or mapping functional relationships there must be a means of quantifying its accuracy. To this end, an algorithm based on statistical theory is developed and presented herein. The approach taken in the algorithm is to follow a binomial experimentation concept in order to establish confidence intervals on the

accuracy of the ANN's approximations. Once a network is trained, using the points in the training set, and an acceptable tolerance level for the approximation error is defined (the error between the exact functional relationship and ANN at any point in the design space), then the problem of network accuracy may be defined in terms of a yes or no question, that is whether the network error at any point in the design space is greater than the specified tolerance level or not. Now, if one randomly selects a number of points in the design (input) space, for every point there would be two possible outcomes to this question. Either the network error, corresponding to the design point, is greater than the tolerance level or it is not. Experiments of this type, wherein repeated independent trials with two possible outcomes are performed, are known as binomial experiments[8].

Assume that n trials have been performed wherein for every trial the network error is simulated for each input point, and the trial is considered a success if the network is greater than the tolerance level, and a failure if it is not. Denote the number of successes in the n trials by the binomial random variable X and the probability of success by p. A point estimator for p is given by $\hat{p} = \frac{X}{n}$. Now, if the unknown probability p is not expected to be too close to zero or one, a confidence interval for p may be established using the distribution of the point estimator $\hat{p}$ through the following theorems.

Theorem 1. If X is a binomial random variable with mean $\mu = np$ and variance $\sigma^2 = npq$, then the limiting form of the distribution of

$$Z = \frac{X - np}{\sqrt{npq}}, \tag{10}$$

as $n \to \infty$, is the standardized normal distribution $n(z; 0, 1)$.

Proof: is provided in ref. 9. The normal distribution is a very good approximation when the sample size n is large and p is close to 0.5. It is even a fairly good approximation for small n so long as p is not very close to 0 or 1. From this theorem, the distribution of $\hat{p}$ is approximately normally distributed with mean, $\mu_{\hat{p}} = E(\frac{X}{n}) = p$, and variance, $\sigma^2_{\hat{p}} = \sigma^2_X/n^2 = pq/n$. Now, a confidence interval for the parameter p is established in the following theorem.

Theorem 2. For $n \geq 30$; a $(1-\alpha)100\%$ confidence interval for the binomial parameter p is approximately

$$\hat{p} - Z_{\alpha/2}\sqrt{\frac{\hat{p}\hat{q}}{n}} < p < \hat{p} + Z_{\alpha/2}\sqrt{\frac{\hat{p}\hat{q}}{n}} \tag{11}$$

where $\hat{p}$ is the proportion of success in a random sample of size n , $\hat{q} = 1 - \hat{p}$, and $Z_{\alpha/2}$ is the value of the standard normal curve leaving an area of $\alpha/2$ to the right.

Proof: is given in ref. 9. If p is the center of a $(1-\alpha)100\%$ interval, then $\hat{p}$ estimates p without error. However, in most cases $\hat{p}$ will not be equal to p , but the size of the difference will be less than $Z_{\alpha/2}\sqrt{\frac{\hat{p}\hat{q}}{n}}$ with $(1-\alpha)100\%$ confidence. It should be noted that only the upper bound expression in Eq. (11) is useful for application to the accurate design of ANN. The size of the sample required to ensure that the error in estimating p by $\hat{p}$ will be less than a number, say e, is given in the following theorem.

Theorem 3. If $\hat{p}$ is used as an estimate of p, one can be $(1-\alpha)100\%$ confident that the error will be less than a specified magnitude e when the sample size is

$$n = \frac{Z^2_{\alpha/2}\hat{p}\hat{q}}{e^2} \leq \frac{Z^2_{\alpha/2}}{4e^2} \tag{12}$$

Proof: is given in ref. 9. It is observed from Eq. (12) that the sample size needed is a function of $\hat{p}$, which itself is computed from the sample. There are two ways around this. One is to take a preliminary small sample with $n_1 \geq 30$ to obtain $\hat{p}$, and use that estimate in Eq. (12) to compute the sample size needed for the desired accuracy. The second option is to use the upper bound expression in Eq. (12), instead of the equality term, which does not depend on $\hat{p}$. However, one must be aware that upper bound expression generally provides conservative results, i.e., it would lead to large values of the required sample size. It is noted that the upper bound expression becomes exact at $\hat{p} = 0.5$.

Now, with aid of these theorems, an algorithm is developed to train feedforward networks with quantified degree of accuracy, given a confidence level. The algorithm is presented in the following steps.

Design Algorithm

a. Define a training set $\{\Delta^0, Y^0_{pm}\}$ to be used as the initial training set for the network design.
b. Choose a feedforward network with one hidden layer. The hidden layer can be any type of layer, such as tan sigmoid, log sigmoid, etc. The output layer should be a pure linear layer.
c. Train the network using the sequential algorithm described earlier. If, however, the optimization does not converge, one has to either increase the limit on the number of epochs or sequences of networks, decrease the desired network error tolerance, or restart the training with a different set of

initial conditions. The initial values of the weights of the ANN before training are the initial conditions.

d. Choose a confidence level α for the network accuracy.

e. Choose a desired tolerance for the network error, e_{tol}. This is the acceptable difference between the ANN's approximation and the exact functional value(s) at any point.

f. Choose a tolerance level, p_{tol}, for the probability of the network exceeding the desired error tolerance.

g. Take m samples of the network error by randomly choosing m points in the input (design) space, $\Delta^1 \equiv \{\delta_1, \delta_2, \ldots, \delta_m\}$ and computing the network error for each of the points. Note that none of m points should be in the initial training set (Δ^0). Moreover, the network errors are computed by simulating the ANN as well as the true function for each of the design points and subtracting one from the other. The sample size m must at the least be greater than or equal to 30, however, and needs to be assigned based on the degree of confidence or accuracy that is desired. Let $e_1, e_2, \ldots, e_m$ denote the sampled network errors.

h. Define trial success as the network error exceeding the desired tolerance, and count the number of successes, n_s, in the m trials above. Note that if no successful event is observed in m trials, additional trials (samples) must be taken, up to a designer-defined limit, until a successful event is observed. Additional discussion is provided on this point later in this section.

i. Compute the sample proportion, $\hat{p}$ and $\hat{q}$

$$\hat{p} = \frac{n_s}{m} \; ; \; \hat{q} = 1 - \hat{p} \tag{13}$$

j. Compute the upper bound confidence level on the probability of network error exceeding e_{tol}

$$p_u = \hat{p} + Z_{\alpha/2}\sqrt{\frac{\hat{p}\hat{q}}{m}} \tag{14}$$

k. if $p_u \leq p_{tol}$, accept the designed network. Otherwise, add the m sample points to the training set, i.e., $\Delta = [\Delta^0 \, \Delta^1]$; $Y_{pm} = [Y^0_{pm} \, Y^1_{pm}]$, and redesign the network by going back to step (a) of the algorithm and repeating the entire algorithm. This procedure may be repeated until convergence is achieved or a limit on the number of iterations, as defined by the designer, is reached. It should be mentioned that the size of the ANN (nodes in the hidden layer) may need to be increased if the learning rate of the network becomes too slow during training or the desired network error tolerance can not be achieved.

As mentioned in step (g) of this algorithm, there is a possibility that no successful event is observed in the m trials. In such a case, one has to take more samples until a successful event is observed. However, a limit should be established on the sample size such that if no successful event is observed the trained ANN is accepted. Such a limit may be established from theorem 3. For example, for a sample size of s, if no successful event is observed, it implies that $\hat{p} < \frac{1}{s}$. Assume that a $(1-\alpha)100\%$ confidence is desired with 1% tolerance on the true probability of success, then from theorem 3, the sample size necessary for this level of tolerance and accuracy is established as follows

$$n = \frac{Z^2_{\alpha/2}\hat{p}\hat{q}}{e^2} \leq \frac{10000 Z^2_{\alpha/2}}{s} \tag{15}$$

Which would be satisfied if

$$s \geq 100 Z_{\alpha/2} \tag{16}$$

For example, for 95% confidence level, $Z_{\alpha/2} = 1.96$, so that the desired network accuracy would be obtained if the sample size is not smaller than 196.

Numerical Example

In order to illustrate the feasibility of the proposed ANN design and training approach, it is used in the design and training of neural networks used in a dynamics and controls analysis application for the NASA's Lewis spacecraft. A structural model of the spacecraft, consisting of the rigid-body modes, and the first ten flexible modes, is used in the analysis. The attitude control system model included full models (as they were available) of reaction wheels, rate gyros, and the star tracker. However, a linearized model of the wheel dynamics was used. A Kalman filter was designed and used to estimate the vehicle's attitude from the sensor data. The reaction wheel dynamics included the linear friction model, limits on the input command voltages and digital voltage quantization, as well as the quantization effects on wheel RPM outputs due to the wheel's optical encoder. To each gyro dynamic model output channel, random signals were added, which represent random drift walk and instrument noise. The modeling of the star tracker included noise and alignment errors. The spacecraft disturbances included environmental disturbances, which included gravity gradient torques, drag torques, magnetic unloading, as well as, a periodic disturbance at 0.3 Hz in roll and yaw axes. Both, the spacecraft structure and the attitude control system were modeled using the various blocks of the SIMULINK software package.

Here, assume that there is uncertainty in the magnitudes of the first two flexible modes, and therefore it

is desired to design a neural network to map the relationship between the changes in the frequencies of those modes as well as changes in the attitude control bandwidth to the dynamic performance of the spacecraft. In this case, the performance is taken as peak-to-peak steady-state response in the pitch axis. A multiplicative scaling variable was used for each of the input variables. The scaling variable associated with the flexible mode frequencies had a range of 0.85 to 1.2 each, and the scaling variable associated with the controller bandwidth had a range of 0.5 to 1.5. With a uniform intervals of 0.05 and 0.2 used for the frequency scaling variables and bandwidth scaling variables, respectively, an initial training set, consisting of 384 training points, was generated. The training data was generated by performing a closed-loop dynamic simulation of the system for each combination of values of the scaling variables, and for the disturbances discussed earlier. Each simulation was a discrete linear simulation, and was run for one orbit, with each orbit assumed to be 5996 seconds in duration. After each simulation run, the peak-to-peak response of the spacecraft in the pitch axis was computed and placed in the appropriate location in the training data output. It was observed from the training data that certain combination of values of scaling variables resulted in dynamic instability, resulting in huge peak-to-peak response levels. These values were all capped at 1000 arc-sec to avoid numerical conditioning problems.

Following the statistics-based, sequential (first variation), algorithm outlined in the paper, a two-layer feedforward network, with a tan sigmoid hidden layer and a pure linear output layer, was designed and trained to provide the desired mapping. The network was designed to have a 99% confidence level that the probability of its approximation exceeding 5% error level would be no greater than 5 percent. The entire training process was performed using 'trainlm' routine of the MATLAB's Neural Network Toolbox, which is based on Levenberg-Marquardt training approach. The history of the training process is provided in Table 1. First, a two-layer feedforward network with six nodes was initialized and trained, with a sum squared error (SSE) goal of 0.0001 for the output normalized data (normalized with respect to maximum absolute value). This network reduced the SSE to 0.0003686 after 400 epochs of training. However, the training of this net was stopped after 400 epochs due to lack of progress in reducing the SSE (less than 0.1% change in 25 epochs). Following the sequential approach, the network error was computed and used as the new training output data for a next net. The number of nodes in the second net was randomly chosen between 3 and 8, and turned out to be 5. The second ANN reduced the SSE to 0.0002824 after 500 epochs of training, at which time the training was stopped due to lack of progress. The procedure continued on, designing and training three more ANNs, before the target SSE was reached, as indicated in Table 1. Now, following the statistics-based approach,

Table 1. Training History, first ANN

ANN No	No. of Nodes	No. of Epochs	SSE
1	6	400	0.0003686
2	5	500	0.0002824
3	4	250	0.0002694
4	5	750	0.0002403
5	7	591	0.0000949
Total	27	2491	

100 points in the feasible range of the variable space were chosen randomly, and then used in the discrete simulation to generate peak-to-peak response values for the system. Next, each point in the test data was also simulated using the neural network trained initially. For each test point, the output of the neural network was compared to the true output (simulation results), which resulted in 19 out of the 100 test points having a network error greater than 5%, the desired accuracy. The proportion from Eq. (13) turns out to be 0.19, which results in the upper bound value for the probability of network exceeding the desired accuracy, from Eq. (14), of 0.2912, for a 99% confidence level. This was a well above the desired tolerance on the probability of failure, which was set at 5 percent. Therefore, the initial ANN was rejected, and the test data was added to the original training data to form the updated training data to be used to train the next network.

Initializing the ANN to be trained at the weights and biases of the first network, a two-layer feedforward network with 27 nodes, the network was trained following the sequential algorithm used in the first. With the SSE goal of 150 (for the actual (non–normalized) output), and using the Levenberg-Marquardt routine, the network reduced the SSE from 1.59e+6 to 232.65, after 800 epochs of training, before training was stopped due to lack of progress. This network reduced the SSE to 149.5 after 400 epochs of training. Following the sequential approach, the network error was computed and used as the new training output data for a next net. The number of nodes in the second net was randomly chosen between 3 and 8, and turned out to be 8. The second ANN reduced the SSE to 149.5, after 585 epochs of training, where at the SSE goal was reached. The over-

all training history for the second network is given in Table 2.

Table 2. Training History, second ANN

ANN No	No. of Nodes	No. of Epochs	SSE
1	27	800	232.65
2	8	585	149.50
Total	35	1385	

Similar to the treatment for the previous network, approach, 100 points in the feasible range of the variable space were chosen randomly, and then used in the discrete simulation to generate peak-to-peak response values for the system. Next, each point in the test data were also simulated using the second neural network. For each test point, the output of the neural network was compared to the true output (simulation results), which resulted in none out of the 100 test points having a network error greater than 5%, the desired accuracy. Although, the proportion from Eq. (13) turns out be null, which results in the upper bound value for the probability of network exceeding the desired accuracy, from Eq. (14), being 0. However, it should be remembered that the upper bound on the probability, as represented by Eq. (14), is not valid at probabilities too close to 0 or 1. However, even if one conservatively assumes that the proportion was at 0.01 (which corresponds to 1 failure in 100 samples), the upper bound value for the probability of network exceeding the desired accuracy becomes 0.0357, which is well below the desired level of 0.05, and thus the network is accepted.

Concluding Remarks

This paper presented a novel methodology for efficient and fast training neural networks, with specified accuracy. Neural networks were considered within the context of dynamics and controls analysis/design to approximate the functional relationships between design change parameters (be they structural or material properties, in the disturbance environment, or in the control system design) and the performance of the system/component. A critical concern with any approximation is its accuracy. Typical neural network training involves the use of a select set of input and output data, taken from the functional relationship to be approximated. If these training points are chosen judiciously, the trained neural network should give a very good approximation. However, there is no guarantee that the neural network will continue to give a good approximation of the relationship for those points not in the training set. The design methodology presented in this paper addressed this problem, in terms of allowing the design of neural networks to a specific level of accuracy, for a given statistical confidence level, and accounting for input and output data not used in the original training set. Specifically, two-layer feedforward neural networks are designed to approximate the functional relationship between the component/spacecraft design changes and measures of its performance. A training algorithm, based on statistical sampling theory, was presented, which guarantees that the trained networks provide a designer-specified degree of accuracy in mapping the functional relationship. Within each iteration of this statistical-based algorithm, a sequential design algorithm was used for the design and training of the feedforward network to provide rapid convergence to the network goals. Here, at each sequence a new network was trained to minimize the error of the previous network. The design algorithm attempts to avoid the local minima phenomenon that hampers the traditional network training, thereby speeding up the training process. Numerical examples carried out on a NASA spacecraft application demonstrated the feasibility of the proposed neural network design methodology.

References

1. K.S. Narendra, "Adaptive Control of Dynamical Systems Using Neural Networks", *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, ed. by D.A. White and D.A. Sofge, Van Nostrand Reinhold, New York, 1992., pp. 141–183.
2. K. Funahashi, "On the Approximate Realization of Continuous Mappings by Neural Networks", *Neural Networks*, Vol. 2, 1989, pp. 183–192.
3. A.R. Gallant and H. White, "There Exists a Neural Network That Does Not Make Avoidable Mistakes", *Proceedings of the IEEE 2nd International Conference on Neural Networks*, 1988, pp. 657–664.
4. S. Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan College Publishing Co., New York, 1994.
5. D.E. Rumelhart and J.L. McClelland, *Parallel Distributed Processing, Vol. 1*, MIT Press, Cambridge, MA, 1986.
6. Hagan, M. T., and Menhaj, M. B., "Training Feedforward Networks with the Marquardt Algorithm", *IEEE Transactions on Neural Networks*, Vol. 5, No. 6, November 1994, pp. 989–993.
7. M.I. Elmasry (ed.), *VLSI Artificial Neural Networks Engineering*, Kluwer Academic Publishers, Norwell, MA, 1994.

8. K. Wawryn and B. Streszewski, "Low Power VLSI Neuron Cells for Artificial Neural Networks", *Proceedings of the 1996 IEEE International Symposium on Circuits and Systems*, 1996, pp. 372–375.

9. R.E. Walpole and R.H. Myers, *Probability and Statistics for Engineers and Scientists*, Macmillan Publishing Co., New York, 1985.