DESIGN OF NEURAL NETWORKS FOR FAST CONVERGENCE AND ACCURACY: DYNAMICS AND CONTROL

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

***Abstract*-A 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, such that once properly trained, they 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 or nonlinear dynamics of the system/components. 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 proposed method should work for applications wherein an arbitrary large source of training data can be generated. Two numerical examples are performed on a spacecraft application in order to demonstrate the feasibility of the proposed approach.**

I. 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

The authors are with the Guidance and Control Branch, NASA Langley Research Center, Hampton, VA, 23681–2199

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 an aerospace system, as the definitions and the designs of the system 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 analysis and 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, once properly trained, can be used to significantly speed up the design and analysis process of aerospace systems by allowing rapid trade analysis as well as quick evaluation of potential impacts of design changes. It should be emphasized here that the training time is not included in this assertion. It is envisioned that the training of networks can be done autonomously during off hours, so that they are made available to the designer/analyst when required. 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 to develop a methodology for efficient and fast training of neural networks, with specified accuracy. Neural network

applications are considered to provide static maps which 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; and to provide dynamic maps which approximate complex and nonlinear dynamics of the system/components. 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. Bayesian statistical regularization [1] and the Cross-Validation technique [2] can provide good approximation for points not in the training set. 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. The proposed method should work for applications wherein an arbitrary large source of training data can be generated. Numerical examples based on a spacecraft application are carried out to demonstrate the feasibility of the proposed 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 a section with a detailed discussion on the convergence behavior of the proposed ANN synthesis methodology and a section with numerical examples of the

proposed approach applied to a NASA spacecraft model. Finally, a concluding remarks section will close the paper.

II. 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 the design phase of the aerospace system almost all components go through changes, with each change having the potential to affect the performance of the system to some degree. In many instances, these changes are expected to affect the performance of the system so much as to warrant a partial or full re-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 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 analyses 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. 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.

III. 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 advantage 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 expected 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., a nonlinear reaction wheel with friction. Dynamic approximations via ANN are achieved by using time delays and feedback of the output back to the ANN input, which is defined as recurrence.

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 input space not included in the training set.

In this paper, a feedforward network with a single hidden layer, like the one shown in Fig. 1, is considered. The feedforward network is designed to map the relationship between an $n_c \times 1$ input vector, δ, and an $n_p \times 1$ output vector y_p. 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 [3]-[5]. 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

$$Y_n = W_2(f(W_1\Delta + B_1)) + B_2. \tag{1}$$

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 is the n_h x q bias matrix, each column is b_1, the bias vector for the hidden layer; B_2 is the n_p x q bias matrix, each column is b_2, the bias vector for the output layer; f represents the activation function for the hidden layer; Δ is a $n_c \times q$ 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. For this work, the parameters of the feedforward network are adjusted using either the back-propagation method [2], [6], or the Levenberg-Marquardt technique [7]. If q sets of points are used for training the network, then the cost function to be minimized, in terms of the sum squared error (SSE) of the network, can be written as

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

where Y_d is a n_p x q matrix of the true outputs.

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 [8]-[9]. 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 selecting 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.

IV. NETWORK TRAINING FOR ACCURACY AND FAST CONVERGENCE

A. *ANN Modeling*

As mentioned earlier, ANNs may be used, within the context of dynamics and control,

to approximate the functional relationship between changes in the system components and the spacecraft performance measures (static maps) or to provide maps which approximate complex and nonlinear dynamics of the system/components. 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 Fig. 2. 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 of the external and internal disturbances, as well as the location (on the structure) where these disturbances are acting. 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 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.

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. The frequency-domain performance measures are typically the stability margins, closed-loop bandwidth, loop shapes, etc. In the static map, an ANN is

sought which can map the relationship between the design change vector δ and the spacecraft performance measure vector y_p, where

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

To do this the ANN must be trained properly using training data consisting of a collection of spacecraft performance measures corresponding to a number of design changes in the range of interest.

In a dynamic map, the ANN is used to approximate the nonlinear and complex dynamics of the spacecraft or its components. For example, ANN can approximate the nonlinear friction of a reaction wheel assembly or the nonlinear large-angle attitude dynamics of the spacecraft, etc. Assume that the dynamics of the system considered for approximation is given by

$$\dot{x} = f(x, u). \tag{4}$$

Then, an ANN is designed to approximate the solution of this system of ordinary differential equations from one time step to the next. The solution of the state vector in Eq. (4) can be effectively approximated by

$$x(t + T) = g(x(t), u(t), u(t + T)) \tag{5}$$

where T denotes a small time increment and $g(.)$ is typically an unknown function which has to be determined computationally by solving the system of ordinary differential equations (Eq. (4)). Note that in some cases, e.g., digital control systems where inputs are sampled and held, the term $u(t + T)$ in Eq. (5) may be dropped. Now, an ANN can be designed to approximate the nonlinear system in Eq. (5), as illustrated in Fig. 3. In this case, the inputs to the network are the external inputs and the state at time *t* (or step *k*) and the output of the network is the state at time *t*+T (or step *k*+1). The training data for the ANN would be a collection of inputs and states in the range of interest as input data, and one-step-ahead propagated state as output data.

B. *Training with Fast Convergence*

Here, an algorithm is presented to help with the three problems associated with the training of ANNs, namely, size of the network, excessive training time, and 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 an 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 (hyperbolic tangent), log sigmoid ($\left(1 + e^{-x}\right)^{-1}$), etc. The output layer should be a pure linear layer.

b. Initially, use a small number of nodes, n_0, for the hidden layer. The algorithm will automatically increase that number as necessary to achieve the required error tolerance.

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 current network number. Note that techniques, such as back-propagation procedure, or 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.

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

$$W_1^F = W_1^1 \; ; \; B_1^F = B_1^1$$

$$W_2^F = W_2^1 \; ; \; B_2^F = B_2^1 \tag{6}$$

$$f^F = f^1$$

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

$$D_1 = Y - \left[W_2^1\left(f^1\left(W_1^1 X + B_1^1\right)\right) + B_2^1\right] \tag{7}$$

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

g. Use the pair $[X, D_1]$ as the new training data. The idea is to train this second network to approximate the network errors computed in Step f. Then, the two networks will be combined into a single network (see Steps k and l) which will reduce the network errors.

h. Choose a new feedforward network with the same architecture as the previous, i.e. one hidden layer and one output layer. Again, 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 remain as a pure linear layer.

i. Use a small 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 of nodes chosen is different from that used in the previous network. This would force the number of design variables (degrees of freedom) to change 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} (from step c), 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

$$W_1^F = \begin{bmatrix} W_1^F \\ W_1^2 \end{bmatrix} \; ; \; B_1^F = \begin{bmatrix} B_1^F \\ B_1^2 \end{bmatrix}$$

$$W_2^F = [W_2^F \quad W_2^2\,] \; ; \; B_2^F = B_2^F + B_2^2 \tag{8}$$

$$f^F = \begin{bmatrix} f^F \\ f^2 \end{bmatrix}$$

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

$$W_1^F \leftarrow \begin{bmatrix} W_1^F \\ W_1^2 \end{bmatrix} \; ; \; B_1^F \leftarrow \begin{bmatrix} B_1^F \\ B_1^2 \end{bmatrix}$$
$$W_2^F \leftarrow [W_2^F \quad W_2^2] \; ; \; B_2^F \leftarrow B_2^F + B_2^2 \tag{9}$$
$$f^F \leftarrow \begin{bmatrix} f^F \\ f^2 \end{bmatrix}$$

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

$$D_2 = D_1 - \left[W_2^2\left(f^2\left(W_1^2 X + B_1^2\right)\right) + B_2^2\right]. \tag{10}$$

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 (e), but using the overall weighting and bias matrices, and activation functions for the initial guess, i.e.,

$$W_1^1 \leftarrow W_1^F \; ; \; B_1^1 \leftarrow B_1^F$$
$$W_2^1 \leftarrow W_2^F \; ; \; B_2^1 \leftarrow B_2^F \tag{11}$$
$$f^1 \leftarrow f^F.$$

The first variation may converge faster, but could lead to larger than required network size. On the other hand, the second variation could lead to substantially longer training time since the size of the network to be trained increases at each step.

C. *Network Accuracy*

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 increases 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 [10].

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 error 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 $\hat{p}$. The distribution of $\hat{p}$ is approximately normally distributed with mean, $\mu_{\hat{p}} = E\left(\frac{X}{n}\right) = p$, and variance, $\sigma^2_{\hat{p}} = \sigma^2_X/n^2 = pq/n$, where $q = 1 - p$ [10]. Now, a confidence interval for the parameter p can be established for a sample of adequate size. For $n \geq 30$; a

$(1-\alpha)100\%$ confidence interval for the binomial parameter p is approximately

$$\hat{p} - Z_{\alpha/2}\sqrt{\frac{\hat{p}\hat{q}}{n}} < p < \hat{p} + Z_{\alpha/2}\sqrt{\frac{\hat{p}\hat{q}}{n}} \tag{12}$$

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 [10]. 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. (12) 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, with $(1-\alpha)100\%$ confidence, is given by [10]

$$n = \frac{Z^2_{\alpha/2}\hat{p}\hat{q}}{e^2} \leq \frac{Z^2_{\alpha/2}}{4e^2}. \tag{13}$$

It is observed from Eq. (13) 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. (13) to compute the sample size needed for the desired accuracy. The second option is to use the upper bound expression in Eq. (13), 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 equations, 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 $\left\{\Delta^0, Y^0\right\}$ 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 or any other training algorithm. 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 weights.

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

e. Choose a desired tolerance for the network point-wise relative error, $\widetilde{e}_{tol}$. This represents the acceptable point-wise relative difference between the ANN's approximation and the exact functional value(s) at any point in the input space (not limited to the training set only). In this paper, this tolerance is defined in a percent form. Note: This error tolerance should not be confused with the error tolerance e_{tol} (on the network SSE), used in the training algorithm (see step c).

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. It is best 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. Let $e_1, e_2, \ldots, e_m$ denote the sampled network errors (percent relative errors). 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.

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 $\hat{q}$

$$\hat{p} = \frac{n_s}{m} \; ; \; \hat{q} = 1 - \hat{p} \tag{14}$$

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

$$p_u = \hat{p} + Z_{\alpha/2}\sqrt{\frac{\hat{p}\hat{q}}{m}} \tag{15}$$

k. If $p_u \leq p_{tol}$, accept the designed network. Otherwise, add the m (or those for which the network error exceed the tolerance) sample points to the training set, i.e., $\Delta = \left[\Delta^0 \; \Delta^1\right]; \; Y = \left[Y^0 \; Y^1\right]$, 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 cannot be achieved.

l. In some applications, the user may desire to have different accuracy requirements based on the absolute or relative magnitude of the input parameters. In such a case, it is more feasible to have multiple stochastic criteria for network accuracy. The prescribed stochastic algorithm may easily be adopted to handle multiple stochastic criteria for network accuracy. Steps (a)-(c) would remain the same. Perform steps (d)-(j) for each required accuracy condition. In

step (k), determine if $p_u \leq p_{tol}$ for each accuracy requirement. If everyone of the accuracy tests pass, accept the network. Otherwise, add the random sample points, either all the random test data or those for which the network error exceed the tolerance, to the training set, i.e.,$\Delta = \left[\Delta^0\, \Delta^1\right];\ Y = \left[Y^0\, Y^1\right]$, and redesign the network by going back to step (a) of the algorithm and repeating the entire algorithm.

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 Eq. (13). 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 Eq. (13), with $\hat{p} = \frac{1}{s}$ and $\hat{q} = 1$, the sample size necessary for this level of tolerance and accuracy is established as follows

$$n = \frac{Z^2_{\alpha/2}\hat{p}\hat{q}}{e^2} \leq \frac{10000 Z^2_{\alpha/2}}{s} \tag{16}$$

which would be satisfied if

$$s \geq 100 Z_{\alpha/2}. \tag{17}$$

For example, for a 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.

V. CONVERGENCE

In this section, convergence behavior of the two key parts of the proposed ANN synthesis algorithm is investigated through empirical analysis. In the empirical approach, convergence of the algorithms is analyzed by investigating the behavior of each algorithm versus the functional

complexity of the map the network has to approximate. The measure of functional complexity used here is the functional variation, which is defined as follows for scalar functions

$$V = \int_{x_1}^{x_2} \left| \frac{dy(x)}{dx} \right| dx \tag{18}$$

where $y(x)$ denotes the functional relationship the ANN is to approximate. First, convergence of the cascading part of the algorithm is considered. To accomplish this, the following scalar function is used for ANN approximation.

$$y(x) = (1 - \alpha) \sin(0.2\pi x) + \alpha \sin(30\pi x) \; ; \; 0 \leq x \leq 1 \tag{19}$$

where α is an arbitrary constant between 0 and 1. At $\alpha = 0$, the function is a slow varying low frequency sine function, with small functional variation V(computed as 1.0). As α is increased the high frequency component increases, and with it so does the functional variation V. For example, at $\alpha = 1$, the functional variation takes a value of 60.

Two variations of the cascading procedure were presented earlier. In the first variation, each new ANN is designed to approximate the network error resulted from its previous network. The second variation is the same as the first except that after each new ANN training the overall network (aggregate of all the cascades) is trained using the original training data. Both variations are considered in the convergence analysis, wherein ANNs were trained for five different values of parameter α. Two-layer feedforward networks, with a tan sigmoid hidden layer and a pure linear output layer, were used. The training set included 50 points, equally distributed between 0 and 1 radians. The performance goal was defined as the sum squared error (SSE) being less than or equal to 1.00e-6. The number of nodes used in the initial network was 20 for $\alpha = 0$; 25 for $\alpha = 0.25$ or $\alpha = 0.5$; and 30 for $\alpha = 0.75$ or $\alpha = 1.0$. These values were chosen in an ad hoc way to take into account the functional complexity of the function to be approximated. The number of nodes used in each of the subsequent network was randomly chosen from the range of 4–11. Tables 1 and 2 summarize the training results for the first and second variations,

respectively.These result indicate that, as expected, the overall size of the network increases as the functional variation increases. Typically, the first variation of the algorithm, wherein each network is trained for the error of the previous network, results in larger network size (more nodes in the hidden layer). However, the difference in the network sizes between the two variation in

Table 1. Convergence of the Cascading Algorithm, First Variation

α	Functional Variation, V	No. of Cascades	Total Number of Nodes
0.00	0.59	0	20
0.25	14.98	2	42
0.50	29.96	7	83
0.75	44.94	4	62
1.00	59.92	4	62

Table 2. Convergence of the Cascading Algorithm, Second Variation

α	Functional Variation, V	No. of Cascades	Total Number of Nodes
0.00	0.59	0	20
0.25	14.98	1	35
0.50	29.96	2	42
0.75	44.94	1	36
1.00	59.92	3	52

most cases is not drastically high. The advantage of first variation over the second is that the size of the network to be trained at each sequence remains fairly small (in this example, between 4 and 11), whereas the size of the network adds up in each sequence of the second variation.

The second part of the ANN synthesis algorithm has to do with the iterative, statistics-based accuracy test for network training. Here, the same empirical approach is used to investigate the convergence of the algorithm. Using the same functional family provided in Eq. (19), the training process described for the cascading part of the algorithm was continued to assess the performance of the statistics-based procedure. The statistical requirements were defined as

desiring 99% confidence that the probability of the relative approximation error exceeding 5% be less than 5 percent. Table 3 provides the results of the statistics-based procedure, using the second variation of the cascading algorithm for network training. Note that the cascading part of the ANN synthesis algorithm is totally distinct from the statistics-based part of the algorithm, hence it should not matter to the statistics-based procedure which is used. In this table, the number

Table 3. Convergence of the Statistics-Based Algorithm

α	No. of Iterations	No. of Additional Training Points	No. of Additional Eval. Points	Final Network Size
0.00	0	0	100	20
0.25	1	7	200	39
0.50	1	7	200	42
0.75	2	5	300	43
1.00	2	6	300	59

of iterations refers to the number of times where the network failed the statistical accuracy test, such that additional training points were used and network was retrained. Here, only those points in the sample that did not meet the accuracy criteria were added to the training set. Note that the initial training of the networks were based on a 50–point training data. It is observed from Table 3 that the required number of iterations and the additional training data are quite reasonable for various levels of functional complexity. Note that in these experiments on a small-scale task, the size of the synthesized networks were quite acceptable. Based on our other experiments, we expect to also obtain reasonably-sized networks for more difficult tasks. Also, it should be noted that even with these empirical results, it is not possible to know how fast the algorithm would converge in general. However, our experience with a variety of problems have demonstrated good convergence behavior.

VI. NUMERICAL EXAMPLES

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. *Spacecraft Application: Static Map*

In the first example, 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, to represent the relative change from the nominal value (e.g., a scaling of

1.1 represents a value 1.1 x nominal value). 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 limited to 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 with a single stochastic accuracy criterion, which was to provide a 99% confidence level that the probability of its approximation exceeding a 5% error level would be no greater than 5%. The entire training process was performed using the 'trainlm' routine of the MATLAB's Neural Network Toolbox, which is based on the 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 4. It should be pointed out that the convergence trends shown in Table 4 were also observed with different initial conditions. Now, following

Table 4. 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	

the statistics-based 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 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. (14) turns out to be 0.19, which results in the upper bound value for the probability of network exceeding the desired accuracy, from Eq. (15), of 0.2912, for a 99% confidence level. This was well above the desired tolerance on the probability of failure, which was set at 5%. Therefore, the initial ANN was rejected, and the test data was added to the original training data.

Initializing the ANN to be trained with 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 run. 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. 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 overall training history for the second network is given in Table 5.

Table 5. 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, 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. (14) turns out be null, which results in the upper bound value for the probability of network exceeding the desired accuracy, from Eq. (15), being 0. However, it should be remembered that the upper bound on the probability, as represented by Eq. (15), 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. Now, with this network the designer can determine the effects of uncertainty in the flexible modes as well as changing the control system bandwidth by an almost instantaneous simulation of the net, bypassing the costly full-up simulations needed in the conventional approach.

B. *Spacecraft Application: Dynamic Map*

A second example, again based on a dynamics and control analysis application for the NASA Lewis spacecraft, is presented. This example illustrates the design and training of an ANN to approximate the rigid-body dynamics of the Lewis spacecraft. Specifically, the ANN will map the functional relationship between the spacecraft's roll, pitch and yaw body rates and applied torques (i.e., disturbance + control) at the kth discrete time step to the roll, pitch and yaw attitude rates at the next discrete time step, k+1. Using the block diagram from Fig. 3, a unit delay is used to feed back the current (kth) attitude rates, which are multiplexed together with the kth applied torques to form the inputs to the ANN.

Previous experience with on-orbit simulations of spacecraft dynamics have shown that the typical attitude rate responses have two distinct levels of magnitude: 1) relatively large levels, typically during the short-duration, initial transient phase; and 2) relatively small, steady-state levels after the initial transients have died down (these steady-state levels are typically several orders of magnitude smaller than the initial transients). In order for a single ANN to approximate these two different levels of responses well, its training set must include adequate numbers of training points from both the large and small magnitude levels or ranges. Moreover, separate

stochastic criterion for network accuracy must be established for the two ranges of attitude rate magnitudes.

For this specific example, a feedforward ANN with a hidden layer with 8 neurons was designed to approximate the rigid-body dynamics. A tan sigmoid activation function was selected for the hidden layer neurons. A target SSE value of 10^{-12} was chosen. The ANN was to have a dual accuracy criteria, such that it provided 99% confidence level that the probability that its approximation of the large magnitude attitude rate responses exceeding a 0.01% error level would be no greater than 5%, and a 99% confidence level that the probability that its approximation of the small magnitude attitude rate responses exceeding a 0.02% error level would also be no greater than 5%. Note that target percent error levels of the ANN outputs, for this second example, were much smaller than the corresponding value (5%) for the ANN design from the first example. The tighter ANN output error levels here were required because of the use of this ANN to approximate a dynamic model (see Fig. 3): because of the recurrent nature of the problem, any errors in the ANN output would be fed back and tend to build up over time, in turn degrading the ANN-based approximation.

The initial training data was generated, in MATLAB/SIMULINK, using a 4th order Runge-Kutta nonlinear integration routine to integrate the following basic rigid body equation of motion:

$$\frac{dw}{dt} = I^{-1}(T - w \times Iw) \tag{20}$$

where w is the vector of spacecraft roll, pitch and yaw attitude rates; I is the spacecraft inertia matrix; and T is the vector of total applied torques to the spacecraft. The initial training set consisted of randomly selected values of the Lewis spacecraft's attitude rates and applied torques, within the expected operating ranges (from both the transient and steady-state ranges), forming the input data. Because the transient phase is much shorter in duration than the steady-state phase, the majority of the initial input points were selected from the latter. To cover the transient phase,

25 random sample points at the large magnitude level were selected, while 200 random sample points at the small magnitude level were selected for the steady-state phase. In addition to these 225 random sample points, various combinations of input training sample points, taken at the upper and lower limits of the expected operational ranges, were included to bring the total number of initial sample points to 354. The corresponding training output, or target, data was computed by treating the input attitude rates as initial values, and integrating Eq. (20) over a single time step interval, for this example, from 0 to 0.256 seconds, while holding the applied torque value constant over that interval. The attitude rates at 0.256 seconds were then recorded as the target data. Prior to ANN training, both the input and output training data were scaled such that they would all lie within a range of +/- 1.

Table 6 summarizes the training history of the ANN. The first column indicates the training iteration for the ANN, with No. 1 being the initial training; the number of training epochs per iteration is shown in the second column ; the third column shows the number of training sample points used in each iteration; the fourth and fifth columns show the computed upper bound on the probability that the ANN output error level exceeding the target error levels for the large and small magnitude responses, respectively; and finally, the last column shows the value of the ANN SSE after each iteration. After the initial training of the ANN, the statistical check, using randomly selected sample points, 104 each from the large and small response magnitude levels, showed that the large magnitude error level upper bound probability was computed to be well within the 5% target, however, the small magnitude error level upper bound probability, at 46%, was well above desired. Following the design algorithm, all 104 random statistical test samples from the small magnitude response levels were added to the original 354 sample points and the ANN trained again. Since the large magnitude error level upper bound probability was already achieved, no additional sample points were added from that response range. This process was repeated through the 7th training iteration, where the random statistical tests showed

that the ANN's small magnitude error level upper bound probability was 7%. Note that, for this example, sequential design of the ANN was unnecessary, and thus the number of hidden layer neurons remained at 8. Although not quite reaching the 5% error level probability and the SSE targets, this ANN was deemed acceptable at that 7% error level probability and 7.3347 x 10^{-9} SSE values.

Table 6. Training History of Rigid Body Dynamics ANN.

Training Iteration	No. of Epochs	No. of Training Sample Points	Large Mag. Upper Bound Prob. (%)	Small Mag. Upper Bound Prob. (%)	SSE
1	4500	354	3.51	46.63	1.4155×10^{-8}
2	4500	458	3.51	20.85	8.5679×10^{-9}
3	1500	562	3.51	19.61	8.3075×10^{-9}
4	1500	666	3.51	23.30	8.1546×10^{-9}
5	1500	770	3.51	19.61	8.0513×10^{-9}
6	1500	874	3.51	14.42	7.5943×10^{-9}
7	1500	978	3.51	7.11	7.3347×10^{-9}
Total	16,500	978			

To test this ANN, it was used in the ANN-based model (depicted in Fig. 3) to represent the rigid-body plant in a SIMULINK simulation of the attitude control system on-board the Lewis spacecraft. Because of the discrete ANN-based model representation of the Lewis dynamics, fixed-step, discrete algebraic updates could be employed to run the simulation. The results from this simulation were then compared with results from another Lewis attitude control SIMULINK simulation, one which used the nonlinear, rigid-body equations (Eq. 20) directly to represent a continuous model of the Lewis spacecraft dynamics. In that simulation, conventional, Runge-Kutta integration was used to propagate the equations of motion. Comparisons between the two simulations have shown that the ANN–based rigid body dynamics model matched the results of

the conventional, differential equation representation very well, and that the ANN-based model simulation executions times were significantly reduced from those of the conventional model simulation times. In this case, the ANN-based simulation took less than one second, while the conventional simulation required at least 2 hours to perform the required analysis/simulation. Fig. 4 shows results from a typical simulation comparison. In this case, both simulations involved a controlled periodic slewing maneuver about the spacecraft's roll axis of +/- 20 degrees (+/– 0.349 radians) during a single orbit. The top plot shows the roll attitude responses from the continuous plant model simulation and the ANN-based model simulation. The bottom plot shows the difference, or error between the simulated roll attitude responses. As can be seen from the two plots, the ANN-based model performed very well.

VII. 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 either 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/components, or to provide dynamic maps that approximate the nonlinear and complex dynamics of the system/components. The design methodology presented in this paper allows the design of neural networks to a specific level of accuracy, for a given statistical confidence level, and accounts for input and output data not used in the original training set. The proposed method should work for applications wherein an arbitrary large source of training data can be generated. Specifically, two-layer feedforward neural networks were 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 might help 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. Finally, it should be pointed out that although the proposed technique has exhibited good convergence properties with reasonably-sized networks for a diverse group of applications, including simple and complex maps, formal proof of its convergence properties is an open issue for future research.

VIII. REFERENCES

[1] F.D. Foresee and M.T. Hagan, "Gauss-Newton Approximation to Bayesian Regularization", *Proceedings of 1997 International Joint Conference on Neural Networks*, Vol. 2, 1997, pp. 1930–1935.

[2] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan College Publishing Co., New York, 1994.

[3] 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.

[4] K. Funahashi, "On the Approximate Realization of Continuous Mappings by Neural Networks", *Neural Networks*, Vol. 2, 1989, pp. 183–192.

[5] 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.D.E.

[6] Rumelhart and J.L. McClelland, *Parallel Distributed Processing, Vol. 1*, MIT Press, Cambridge, MA, 1986.

[7] 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.

[8] M.I. Elmasry (ed.), *VLSI Artificial Neural Networks Engineering*, Kluwer Academic Publishers, Norwell, MA, 1994.

[9] 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.

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

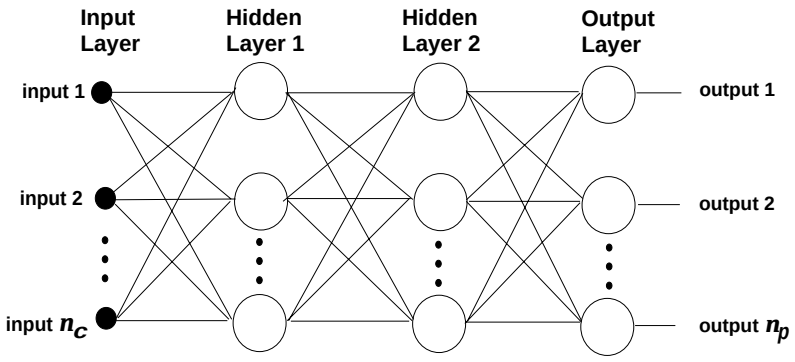

Figure 1. Feedforward ANN.

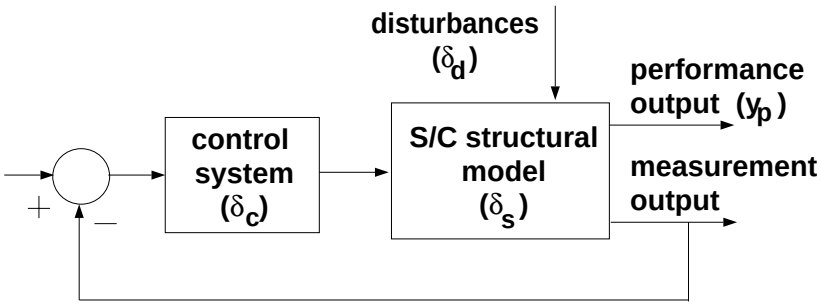

Figure 2. Typical block diagram of controlled spacecraft.

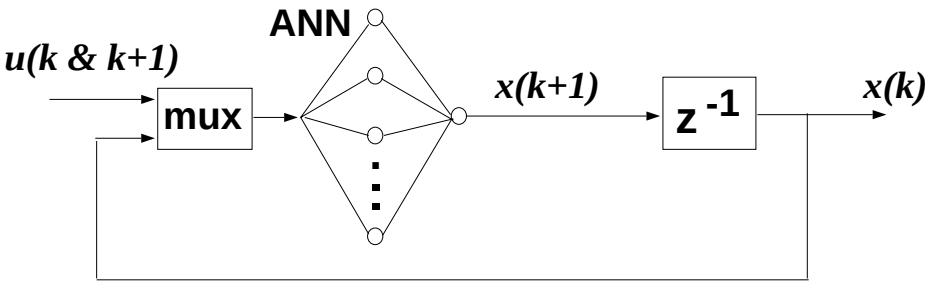

Figure 3. Discrete ANN-based dynamics model block diagram.

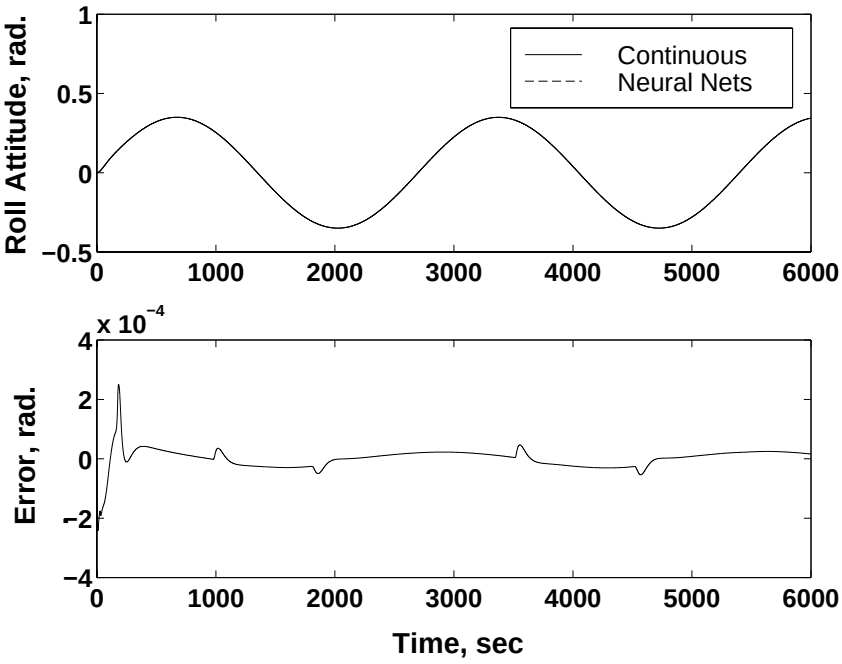

Figure 4. Simulated roll attitude time response and error plots (Neural Net Overlaps Continuous Graph).

P.G. Maghami and D.W. Sparks