Smooth Function Approximation Using Neural Networks

Silvia Ferrari, *Member, IEEE,* and Robert F. Stengel, *Fellow, IEEE*

***Abstract*—An algebraic approach for representing multidimensional nonlinear functions by feedforward neural networks is presented. In this paper, the approach is implemented for the approximation of smooth batch data containing the function's input, output, and possibly, gradient information. The training set is associated to the network adjustable parameters by nonlinear weight equations. The cascade structure of these equations reveals that they can be treated as sets of linear systems. Hence, the training process and the network approximation properties can be investigated via linear algebra. Four algorithms are developed to achieve exact or approximate matching of input–output and/or gradient-based training sets. Their application to the design of forward and feedback neurocontrollers shows that algebraic training is characterized by faster execution speeds and better generalization properties than contemporary optimization techniques.**

***Index Terms*—Algebraic, function approximation, gradient, input–output, training.**

I. Introduction

ALGEBRAIC training is a novel approach for approximating multidimensional nonlinear functions by feedforward neural networks based on available input, output, and, possibly, gradient information. The problem of determining the analytical description for a set of data arises in numerous sciences and applications, and can be referred to as data modeling or system identification. Neural networks are a convenient means of representation because they are universal approximators that can learn data by example [1] or reinforcement [2], either in batch or sequential mode. They can be easily trained to map multidimensional nonlinear functions because of their parallel architecture. Other parametric structures, such as splines and wavelets, have become standard tools in regression and signal analysis involving input spaces with up to three dimensions [3]–[6]. However, much of univariate approximation theory does not generalize well to higher dimensional spaces [7]. For example, the majority of spline-based solutions for multivariate approximation problems involve tensor product spaces that are highly dependent on the coordinate system of choice [8]–[10].

Neural networks can be used effectively for the identification and control of dynamical systems, mapping the input–output representation of an unknown system and, possibly, its control law [11], [12]. For example, they have been used in combination with an estimation-before-modeling paradigm to perform online identification of aerodynamic coefficients [13]. Derivative information is included in the training process, producing smooth and differentiable aerodynamic models that can then be used to design adaptive nonlinear controllers. In many applications, detailed knowledge of the underlying principles is available and can be used to facilitate the modeling of a complex system. For example, neural networks can be used to combine a simplified process model (SPM) with online measurements to model nutrient dynamics in batch reactors for wastewater treatment [14]. The SPM provides a preliminary prediction of the behavior of nutrient concentrations. A neural network learns how to correct the SPM based on environmental conditions and on concentration measurements. The problem of function approximation also is central to the solution of differential equations. Hence, neural networks can provide differentiable closed-analytic-form solutions that have very good generalization properties and are widely applicable [15]. In this approach, a fixed function is used to satisfy the boundary conditions, and a neural network is used to solve the minimization problem subject to the former constraint.

Typically, training involves the numerical optimization of the error between the data and the actual network's performance with respect to its adjustable parameters or weights. Considerable effort has gone into developing techniques for accelerating the convergence of these optimization-based training algorithms [16]–[18]. Another line of research has focused on the mathematical investigation of networks' approximation properties [19]–[22]. The latter results provide few practical guidelines for implementing the training algorithms, and they cannot be used to evaluate the properties of the solutions obtained by numerical optimization. The algebraic training approach provides a unifying framework that can be exploited both to train the networks and to investigate their approximation properties. Both aspects are simplified by formulating the nonlinear representation problem in terms of *weight equations*. The data are associated to the adjustable parameters by means of neural network input–output and, possibly, gradient equations. This translates into a set of nonlinear, transcendental weight equations that describe both the training requirements and the network properties. However, the cascade structure of these equations allows the nonlinearity of the hidden nodes to be separated from the linear operations in the input and output layers, such that the weight equations can be treated as sets of algebraic systems, while maintaining their original functional form. Hence, the nonlinear training process and

Manuscript received August 6, 2001; revised October 15, 2003. This work was supported by the Federal Aviation Administration and the National Aeronautics and Space Administration under FAA Grant 95-G-0011.

S. Ferrari is with the Department of Mechanical Engineering and Materials Science, Duke University, Durham, NC 27708 USA (e-mail: Sferrari@duke.edu).

R. F. Stengel is with the Department of Mechanical and Aerospace Engineering, Princeton University, Princeton, NJ 08544 USA.

Digital Object Identifier 10.1109/TNN.2004.836233

related approximation properties can be investigated via linear algebra.

In this paper, smooth multidimensional nonlinear functions are modeled using an algebraic approach. Depending on the design objectives, algebraic training can achieve exact *or* approximate matching of the data at the training points, with *or* without derivative information. Its advantages with respect to optimization-based techniques are reduced computational complexity, faster execution speeds, and better generalization properties. Furthermore, algebraic training can be used to find a direct correlation between the number of network nodes needed to model a given data set and the desired accuracy of representation. For example, it is shown that a set of p control-system gains can be matched exactly by a sigmoidal network with p nodes, and that its weights can be determined by solving algebraic equations in one step. Four algebraic training algorithms are developed and demonstrated by training a *forward neural network* that models the set of equilibria (or trim map) of a transport aircraft and a *feedback neural network* that models a nonlinear control system (implemented in [23] and [24]).

Algorithms that determine exact solutions (presented in Section IV-A and D) are valuable for incorporating precise knowledge of a system in the neural networks that represent it. In many applications (e.g., [13]–[15], [23]), this information is available *a priori* and can be complemented by posterior data. In such cases, the objective is not to spare the number of nodes, but rather to produce a network with sufficient degrees of freedom while retaining good generalization properties, as accomplished in the examples presented in Section V-A and C. In other applications (e.g., [25]), the objective is to synthesize a large data set by a parsimonious network. Then, the approximate-solution algorithm presented in Section IV-C can be used, as shown in Section V-B. In this paper, the algebraic approach is applied to the batch training of feedforward sigmoidal networks for the modeling of noise-free data. Work in progress and the preliminary results in [26] and [27] show that the approach also has value in analyzing other architectures and in training networks online (i.e., in sequential mode).

II. Development of Neural Network Weight Equations

The set of nonlinear weight equations that relates the neural network's adjustable parameters to the data is obtained by imposing the training requirements on the network's output and gradient equations. Algebraic training is based on the key observation that if all inputs to the sigmoidal functions are known, then the weight equations become algebraic and, often, linear. These inputs are referred to as *input-to-node values*, and they determine the saturation level of each sigmoid at a given data point. The particular structure of the weight equations allows the designer to analyze and train a nonlinear neural network by means of linear algebra, partly by controlling the distribution and saturation level of the active nodes which determine the network generalization properties.

The objective is to approximate a smooth scalar function of q inputs $h : \Re^q \rightarrow \Re$ using a feedforward sigmoidal network of the type shown in Fig. 1. The approach also can be extended to include vector-output functions. Typically, the function to be approximated is not known analytically, but a precise set of input–output samples $\{\mathbf{y}^k, u^k\}_{k=1,\ldots,p}$ can be generated such that $u^k = h(\mathbf{y}^k)$, for all values of k. This set of samples is referred to as *training set*. For example, a high-dimensional partial differential equation solver could be used to compute a smooth fluid flow, and it might be desired to represent the flow field by a neural network. Then, the p training points for the neural network could be derived from the flow solution. The use of derivative information during training can improve upon the network's generalization properties [13]. Therefore, if the partial derivatives of the function $h(\cdot)$ are known with respect to e of its inputs

$$\mathbf{c}^k \equiv \left[\left.\frac{\partial h}{\partial y_1}\right|_{\mathbf{y}^k} \quad \cdots \quad \left.\frac{\partial h}{\partial y_e}\right|_{\mathbf{y}^k} \right]^T, \quad e \leq q, \quad k = 1, \ldots, p \tag{1}$$

they also can be incorporated in the training set: $\{\mathbf{y}^k, u^k, \mathbf{c}^k\}_{k=1,\ldots,p}$.

The scalar output of the network z is computed as a nonlinear transformation of the weighted sum of the input $\mathbf{p}$ and a bias $\mathbf{d}$, plus an output bias b

$$z = \mathbf{v}^T \boldsymbol{\sigma}[\mathbf{W}\mathbf{p} + \mathbf{d}] + b \tag{2}$$

$\boldsymbol{\sigma}[\cdot]$ is composed of sigmoid functions, such as $\sigma(n) = (e^n - 1)/(e^n + 1)$, evaluated at all input-to-node variables n_i with $i = 1, \ldots, s$

$$\boldsymbol{\sigma}[\mathbf{n}] \equiv [\sigma(n_1) \ \cdots \ \sigma(n_s)]^T \tag{3}$$

and

$$\mathbf{n} = \mathbf{W}\mathbf{p} + \mathbf{d} = [n_1 \ \cdots \ n_s]^T \tag{4}$$

where $\mathbf{W}$ and $\mathbf{v}$ contain the input and output weights, respectively. Together with $\mathbf{d}$ and b they constitute the adjustable parameters of the network.

The order of differentiability of (2) is the same as that of the activation function, $\sigma(\cdot)$. Given that the chosen sigmoid functions are infinitely differentiable, the derivative of the network output with respect to its inputs is

$$\frac{\partial z}{\partial p_j} = \sum_{i=1}^{s} \frac{\partial z}{\partial n_i}\frac{\partial n_i}{\partial p_j} = \sum_{i=1}^{s} v_i \sigma'(n_i) w_{ij}, \quad j = 1, \ldots, q \tag{5}$$

where $\sigma'(\cdot)$ denotes the derivative of the sigmoid function with respect to its scalar input. w_{ij} denotes the element in the ith row and the jth column of the matrix $\mathbf{W}$, and it represents the interconnection weight between the jth input and the ith node of the network. Equations (2) and (5) constitute the network's output and gradient equations, respectively. The training requirements are obtained from the training set, as explained in the following paragraphs.

The computational neural network matches the input–output training set $\{\mathbf{y}^k, u^k\}_{k=1,\ldots,p}$, exactly if, given the input $\mathbf{y}^k$, it produces u^k as the output

$$z(\mathbf{y}^k) = u^k, \quad k = 1, \ldots, p. \tag{6}$$

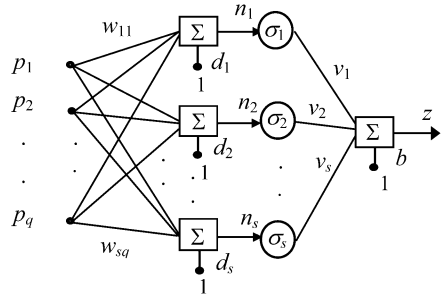

Fig. 1. Sample scalar-output network with q-inputs and s-nodes in the hidden layer.

This is equivalent to stating that the neural adjustable parameters must satisfy the following nonlinear equations:

$$u^k = \mathbf{v}^T \boldsymbol{\sigma}[\mathbf{W}\mathbf{y}^k + \mathbf{d}] + b, \quad k = 1, \ldots, p \tag{7}$$

which are referred to as *output weight equations*. When all the known output elements from the training set are grouped in a vector

$$\mathbf{u} = [\, u^1 \;\; \cdots \;\; u^k \,]^T. \tag{8}$$

Equation (7) can be written using matrix notation

$$\mathbf{u} = \mathbf{S}\mathbf{v} + \mathbf{b} \tag{9}$$

where $\mathbf{b}$ is an s-dimensional vector composed of the scalar output bias b. $\mathbf{S}$ is a matrix of sigmoid functions evaluated at input-to-node values n_i^k, each representing the magnitude of the input-to-node variable n_i to the ith node for the training pair k

$$\mathbf{S} \equiv \begin{bmatrix} \sigma\left(n_1^1\right) & \sigma\left(n_2^1\right) & \ldots & \sigma\left(n_s^1\right) \\ \sigma\left(n_1^2\right) & \sigma\left(n_2^2\right) & \ldots & \sigma\left(n_s^2\right) \\ \vdots & \vdots & \ddots & \vdots \\ \sigma\left(n_1^p\right) & \sigma\left(n_2^p\right) & \ldots & \sigma\left(n_s^p\right) \end{bmatrix}. \tag{10}$$

The nonlinearity of the output weight equations arises purely from these sigmoid functions.

Exact matching of the function's derivatives (1) is achieved when the neural network's gradient evaluated at the input $\mathbf{y}^k$ equals $\mathbf{c}^k$, i.e.,

$$\left.\frac{\partial z}{\partial p_j}\right|_{\mathbf{y}^k} = c_j^k, \quad j = 1, \ldots, e, \quad k = 1, \ldots, p. \tag{11}$$

Hence, the adjustable parameters must satisfy the following *gradient weight equations*

$$\mathbf{c}^k = \mathbf{W}_e^T \{\mathbf{v} \otimes \boldsymbol{\sigma}'[\mathbf{n}^k]\}, \quad k = 1, \ldots, p \tag{12}$$

that are obtained by imposing the requirements in (11) on (5). The symbol "$\otimes$" denotes element-wise vector multiplication. $\mathbf{W}_e$ represents the first e columns of $\mathbf{W}$ containing the weights associated with inputs p_1 through p_e. *Input-to-node weight equations* are obtained from the arguments of the nonlinear sigmoidal functions in (7) and (12)

$$\mathbf{n}^k = \mathbf{W}\mathbf{y}^k + \mathbf{d}, \quad k = 1, \ldots, p \tag{13}$$

where $\boldsymbol{\sigma}'[\,\cdot\,]$ is a vector-valued function whose elements consist of the function $\sigma'(\,\cdot\,)$ evaluated component-wise at each element of its vector argument

$$\boldsymbol{\sigma}'[\mathbf{n}] \equiv [\, \sigma'(n_1) \;\; \ldots \;\; \sigma'(n_s) \,]^T. \tag{14}$$

Equation (12) can be written as

$$\mathbf{c}^k = [\mathbf{B}^k \mathbf{W}_e]^T \tag{15}$$

with the matrix

$$\mathbf{B}^k \equiv \left[v_1 \sigma'\left(n_1^k\right) \;\; v_2 \sigma'\left(n_2^k\right) \;\; \ldots \;\; v_s \sigma'\left(n_s^k\right) \right] \tag{16}$$

explicitly containing only sigmoid functions and output weights.

Since the weight equations relate the neural parameters to the training set, they can be used to investigate the approximation properties of the neural network and to compute its parameters. If the derivative information $\mathbf{c}^k$ is not available, the output weight equations are considered and (15) is ignored. Conversely, if the output information $\mathbf{u}^k$ is not available, (9) is ignored. If all input-to-node values $\{n_i^k\}$ are known, the nonlinear transcendental weight equations (9) and (15) are both algebraic and linear. Based on this assumption, the sigmoidal matrix in (10) is known, and the output weight (9) can be solved for the output weights $\mathbf{v}$. Then, all of the $\mathbf{B}^k$ matrices are known, and the gradient weight (15) can be solved for the input weights $\mathbf{W}_e$. The following section presents four algebraic algorithms that determine the input-to-node values and, then, compute the weights from the linear systems in (9) and (15). Their effectiveness is demonstrated through the examples in Section V.

III. Algebraic Training Algorithms

A. Exact Matching of Function Input–Output Data

Assume that the training set takes the form $\{\mathbf{y}^k, u^k\}_{k=1,\ldots,p}$. Equation (9) admits a unique solution if and only if $\text{rank}(\mathbf{S} \,|\, \mathbf{u}) = \text{rank}(\mathbf{S}) = s$, where $\text{rank}(\,\cdot\,)$ represents the rank of the matrix (e.g., see [28]). Under the assumption of known input-to-node values, $\mathbf{S}$ is a $p \times s$ known matrix. When the number of nodes s is chosen equal to the number of training pairs p, $\mathbf{S}$ is square. If it also is nonsingular and the training data are consistent, (9) is a full-rank linear system for which a unique solution always exists. The input parameters affect the solution of the output weight equations only through the input-to-node values determining the nature of $\mathbf{S}$. Thus, the required weights are not unique. They need be chosen only to assure that $\mathbf{S}$ is full rank. With suitable $\mathbf{S}$, the fit is determined by specifying $\mathbf{v}$ and $\mathbf{d}$ alone.

A strategy for producing a well-conditioned $\mathbf{S}$ consists of generating the input weights according to the following rule:

$$w_{ij} = fr_{ij} \tag{17}$$

where r_{ij} is chosen from a normal distribution with zero mean and unit variance that is obtained using a random number generator. f is a user-defined scalar that can be adjusted to obtain input-to-node values that do not saturate the sigmoids, as explained further below. The input bias $\mathbf{d}$ is computed to center each sigmoid at one of the training pairs $\{\mathbf{y}^k, u^k\}$ from (13), setting $n_i^k = 0$ when $i = k$

$$\mathbf{d} = -\text{diag}(\mathbf{Y}\mathbf{W}^T) \tag{18}$$

$\mathbf{Y}$ is a matrix composed of all the input vectors in the training set

$$\mathbf{Y} \equiv [\mathbf{y}^1 \ \cdots \ \mathbf{y}^k]^T. \tag{19}$$

The "diag" operator extracts the diagonal of its argument (a square matrix) and reshapes it into a column vector. Equation (18) distributes the sigmoids across the input space, as also is suggested by the Nguyen–Widrow initialization algorithm [29]. Finally, the linear system in (9) is solved for $\mathbf{v}$ by inverting $\mathbf{S}$

$$\mathbf{v} = \mathbf{S}^{-1}(\mathbf{u} - \mathbf{b}). \tag{20}$$

In this case, the output bias b is an extra variable; thus, the vector $\mathbf{b}$ can be set equal to zero.

The input elements $\mathbf{y}^k$ from the training set can be normalized. Alternatively, the factor f alone can be used to scale the distribution of the input-to-node values, establishing their order of magnitude. While p of the sigmoids in (10) are centered, the remaining ones come close to being saturated for inputs whose absolute value is greater than 5. Thus (for the chosen sigmoid functions), input-to-node values of order $O(10)$ allow a good fraction of the sigmoids to be highly saturated, contributing to a smooth approximating function and producing a nonsingular $\mathbf{S}$. If (17) has produced an ill-conditioned $\mathbf{S}$, this computation simply is repeated before proceeding to solve (9) (typically, one computation suffices). This algorithm is illustrated by the solid line elements of the flowchart in Fig. 2, and the respective code implementation is shown in [24]. The dashed lines represent modifications to incorporate derivative information, as derived in the next section. The technique is applied in Section V-A to model the longitudinal trim map of an aircraft.

B. Approximate Matching of Gradient Data in Algebraic Training

Exact matching of both input–output and gradient information $\{\mathbf{y}^k, u^k, \mathbf{c}^k\}_{k=1,\ldots,p}$ is achieved when the output and gradient weight (9) and (15) are solved simultaneously for the neural parameters. It is possible to solve both equations exactly when the dimension $(q - e)$ equals p, or when the training set has the special form to be discussed in Section IV-D. In

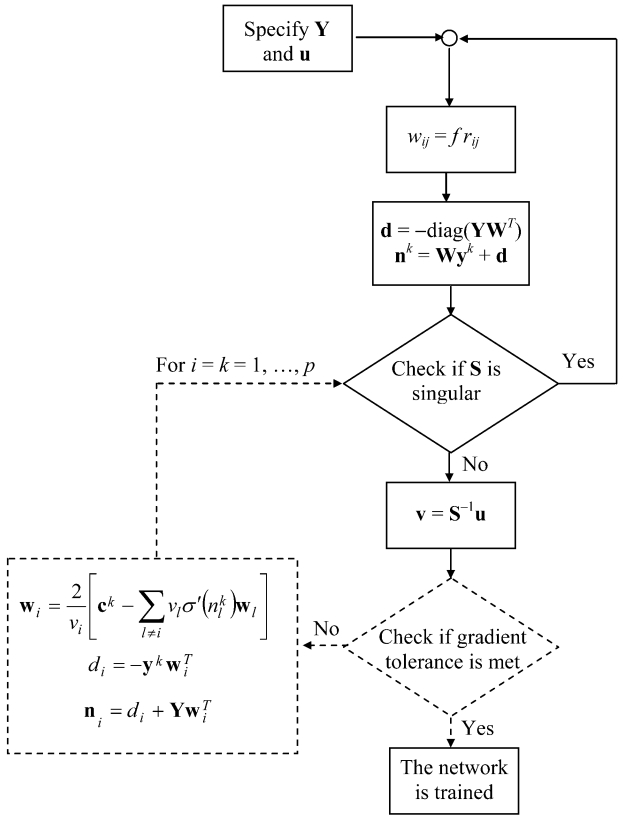

Fig. 2. Exact input–output-based algebraic algorithm with added p-steps for incorporating gradient information.

general, a suitable way to incorporate the gradient equations in the training process is to use (15) to obtain a more stringent criterion of formation for the input weights. The approach of Section IV-A has proven that there exists more than one p-node network capable of fitting input–output information exactly. Using derivative information during training is one approach to choosing the solution that has the best generalization properties among these networks.

A first estimate of the output weights $\mathbf{v}$ and of the input-to-node values n_i^k to be used in (15) can be obtained from the solution of the output equations (9) based on the randomized $\mathbf{W}$ (17). This solution already fits the input–output training data. The input weights and the remaining parameters can be refined to more closely match the known gradients using a p-step node-by-node update algorithm. The underlying concept is that the input bias d_i and the input-to-node values associated with the ith node

$$\mathbf{n}_i \equiv \left[n_i^1 \ \cdots \ n_i^p\right]^T \tag{21}$$

can be computed solely from the input weights associated with it

$$\mathbf{w}_i \equiv \left[w_{i1} \ \cdots \ w_{iq}\right]^T. \tag{22}$$

At each step, the ith sigmoid is centered at the kth training pair through the input bias d_i, i.e., $n_i^k = 0$, when $i = k$. The kth

gradient equations are solved for the input weights associated with the ith node, i.e., from (15)

$$v_i\sigma'\left(n_i^k\right)\mathbf{w}_i = \mathbf{c}^k - \begin{bmatrix} \sum_l v_l\sigma'\left(n_l^k\right)w_{l1} \\ \vdots \\ \sum_l v_l\sigma'\left(n_l^k\right)w_{lq} \end{bmatrix}$$
$$l = 1,\ldots,(i-1),(i+1),\ldots,p \text{ and } l \neq i. \quad (23)$$

The remaining variables are obtained from the initial estimate of the weights. The ith input bias is computed individually

$$d_i = -\mathbf{y}^k\mathbf{w}_i \quad (24)$$

and p of the input-to-node values are updated

$$\mathbf{n}_i = d_i + \mathbf{Y}\mathbf{w}_i. \quad (25)$$

At the end of each step, (9) is solved for a new value of $\mathbf{v}$, based on the latest input-to-node values.

The gradient equations are solved within a user-specified gradient tolerance. At each iteration, the error enters through $\mathbf{v}$ and through the input weights to be adjusted in later steps w_{lj} with $l = (i+1),\ldots,p$. The basic idea is that the ith node input weights mainly contribute to the kth partial derivatives $\mathbf{w}_i$, because the ith sigmoid is centered at $i = k$ and $\mathbf{v}$ can be kept bounded for a well-conditioned $\mathbf{S}$. As other sigmoids approach saturation, their slopes approach zero, making the error associated with w_{lj} smaller. If the gradient with respect to some inputs is unknown, the corresponding input weights can be treated similarly to the input bias. In the limit of p "free" inputs, all weight equations can be solved exactly for the network's parameters. The flowchart in Fig. 2 shows how the input–output-based algorithm can be modified by the p-operations in the dashed box (a code implementation also is shown in [24]). The gradient tolerance can be checked at every step so that the algorithm can terminate as soon as the desired tolerance is met, even if $k < p$. The effectiveness of this algorithm is demonstrated in Section V-A by training a neural network to approximate a longitudinal aircraft trim map based on gradient and input–output information.

C. Approximate Matching of Function Input–Output Data

The algebraic approach can be used to obtain an approximate parsimonious network when the number of training pairs p is large. Section IV-A showed how exact matching of an input–output training set $\{\mathbf{y}^k, u^k\}_{k=1,\ldots,p}$ can be achieved by choosing a number of nodes s that equals p. An exact solution also could be obtained using fewer nodes than there are training pairs, i.e., $s < p$, provided the rank condition $\text{rank}(\mathbf{S}\,|\,\mathbf{u}) = \text{rank}(\mathbf{S}) = s$ is satisfied. These results reflect intrinsic properties of neural networks that are independent of the algebraic approach, and they provide guidelines for the training procedure. For example, when the linear system in (9) is not square ($s \neq p$), an inverse relationship between $\mathbf{u}$ and $\mathbf{v}$ can be defined using the *generalized inverse* or *pseudoinverse matrix* [24]. Typically, (9) will be overdetermined, with more equations than there are unknowns, and its solution will be given by

$$\mathbf{v} = (\mathbf{S}^T\mathbf{S})^{-1}\mathbf{S}^T\mathbf{u} = \mathbf{S}^{\text{PI}}\mathbf{u} \quad (26)$$

where $\mathbf{S}^{\text{PI}}$ constitutes the left pseudoinverse, and $\mathbf{b}$ is set equal to zero for simplicity. If the equations are consistent, (26) provides the exact value for $\mathbf{v}$. If they are not consistent, $\text{rank}(\mathbf{S}\,|\,\mathbf{u}) \neq \text{rank}(\mathbf{S})$, the system in (9) has no solution. In the latter case, (26) provides the estimate that minimizes the mean-square error (MSE) in the estimate of $\mathbf{v}$ and can be used to obtain an approximate solution for the output weight equations.

Whenever a neural network is trained by a conventional algorithm that does not achieve exact matching, such as backpropagation [30], the corresponding weight equations fall into the approximate case above. This is because, given a training set, corresponding weight equations can be written for any network whose parameters constitute either an exact or an approximate solution to these equations. Letting $\hat{\mathbf{u}}$ denote the best approximation to $\mathbf{u}$ obtained from the final neural parameters, the following holds:

$$\hat{\mathbf{u}} = \mathbf{S}\mathbf{v}. \quad (27)$$

Regardless of how the actual network output weight vector $\mathbf{v}$ has been determined, it satisfies (27) along with the actual value of $\mathbf{S}$. Equation (27) minimizes the error $(\mathbf{u} - \hat{\mathbf{u}})$, which is the same error minimized by conventional optimization-based training algorithms [30]. This observation completes the picture by showing how the algebraic approach can deal with the case of $s < p$, typically found in the neural network literature. More importantly, it can be exploited to develop approximate techniques of solution that are computationally more efficient than the conventional iterative methods, such as the one outlined below and implemented in Section V-B.

Based on these ideas, an algebraic technique that superimposes many networks into one is developed. Suppose a neural network is needed to approximate a large training set [i.e., $p \sim O(10^5)$] using a parsimonious number of nodes, s. Conventional methods, such as Levenberg–Marquardt (LM) and resilient backpropagation (RPROP) [31], [32], can successfully train networks with $s < p$, minimizing the error $(\mathbf{u} - \hat{\mathbf{u}})$, but they quickly run out of memory if a large set is used at once in what is referred to as *batch training*. If the training set is divided into smaller subsets, training becomes even more challenging, as the neural network is likely to forget previously learned subsets while it is being trained with new ones. Furthermore, these difficulties are exacerbated by the problem of finding the appropriate number of nodes. On the other hand, when a small subset is used, batch training can be very effective. Many of the conventional algorithms converge rapidly, and the network generalization abilities can be optimized by finding the "best" number of nodes through a trial-and-error procedure.

The technique described here algebraically superimposes networks that individually map the nonlinear function h: $\Re^q \rightarrow \Re$ over portions of its input space into one network that models h over its entire input space. The full training set $\{\mathbf{y}^k, u^k\}_{k=1,\ldots,p}$, covering the full range of the h input space, is divided into m subsets

$$\{\mathbf{y}^k, u^k\}_{k=1,\ldots,p_1}, \{\mathbf{y}^k, u^k\}_{k=p_1+1,\ldots,p_2}, \ldots, \{\mathbf{y}^k, u^k\}_{k=p_{m-1}+1,\ldots,p_m}$$

where $p_m = p$. Each subset is used to train a sigmoidal neural network of the type shown in Fig. 1, whose parameters are indexed by g, where $g = 1, \ldots, m$. That is, each s_g-node network, or *subnetwork*, models the gth subset $\{\mathbf{y}^k, u^k\}_{k=p_{g-1}+1,\ldots,p_g}$ using the weights $\mathbf{W}_g, \mathbf{d}_g$, and $\mathbf{v}_g (b_g = 0)$, and $s_g < p_g$. Then, as suggested by the schematic in Fig. 3, the m networks are *superimposed* to form a s-node network that models the full training set using the weights $\mathbf{W}, \mathbf{d}$, and $\mathbf{v}$, and $s = s_1 + \cdots + s_m$. Fig. 3 shows the equivalence between the group of subnetworks and the network obtained by their superposition. Here, the summation symbols are omitted for simplicity.

The output weight equations of each subnetwork fall into the approximate case described above. Therefore, the gth neural network approximates the vector $\mathbf{u}_g \equiv [u^{p_{g-1}+1} \ \cdots \ u^{p_g}]^T$ by the estimate

$$\hat{\mathbf{u}}_g = \mathbf{S}_g \mathbf{v}_g \tag{28}$$

where $\mathbf{v}_g$ is the actual output weight vector and $\text{rank}(\mathbf{S}_g | \hat{\mathbf{u}}_g) = \text{rank}(\mathbf{S}_g) = s_g$. The input weights of the m networks are preserved in the full input weight matrix

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_1 \\ \vdots \\ \mathbf{W}_m \end{bmatrix} \tag{29}$$

and input bias vector

$$\mathbf{d} = \begin{bmatrix} \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_m \end{bmatrix}. \tag{30}$$

Then, for the full network the matrix of input-to-node values defined as $\mathbf{N} \equiv \{n_i^k\}$, with the n_i^k element in the ith column and kth row, contains the input-to-node value matrices for the m sub-networks along its main diagonal

$$\mathbf{N} = \begin{bmatrix} \mathbf{N}_1 & \ldots & \mathbf{N}_{1m} \\ \vdots & \ddots & \vdots \\ \mathbf{N}_{m1} & \ldots & \mathbf{N}_m \end{bmatrix}. \tag{31}$$

From (13), it can be easily shown that the off-diagonal terms, such as $\mathbf{N}_{1m}$ and $\mathbf{N}_{m1}$, are columnwise linearly dependent on the elements in $\mathbf{N}_1, \mathbf{N}_2, \ldots$, and $\mathbf{N}_m$, so $r(\mathbf{N}) = r(\mathbf{N}_1) + \cdots + r(\mathbf{N}_m) = s_1 + \cdots + s_m = s$. Also, it is found that in virtually all cases examined $\text{rank}(\mathbf{S}) = \text{rank}(\mathbf{N})$. Although a rigorous proof cannot be provided because of the nonlinearity of the sigmoid function, typically $\text{rank}(\mathbf{S}) = s$.

Finally, the output weight equations are used to compute the output weights that approximate the full training set

$$\mathbf{v} = \mathbf{S}^{\text{PI}} \left[\hat{\mathbf{u}}_1^T \quad \cdots \quad \hat{\mathbf{u}}_m^T\right]^T. \tag{32}$$

Because $\mathbf{S}$ was constructed to be of rank s, the rank of $(\mathbf{S} | [\hat{\mathbf{u}}_1^T \ \cdots \ \hat{\mathbf{u}}_m^T])$ is s or, at most, $s + 1$, bringing about a zero or small error during the superposition. More importantly, because the error does not increase with m, several subnetworks can be algebraically superimposed to model one large training set using a parsimonious number of nodes. In practice, the vector $[\hat{\mathbf{u}}_1^T \ \cdots \ \hat{\mathbf{u}}_m^T]$ in (32) can be substituted by the vector $[\mathbf{u}_1^T \ \cdots \ \mathbf{u}_m^T]$ that is directly obtained from the training set and, effectively, contains the output values to be approximated.

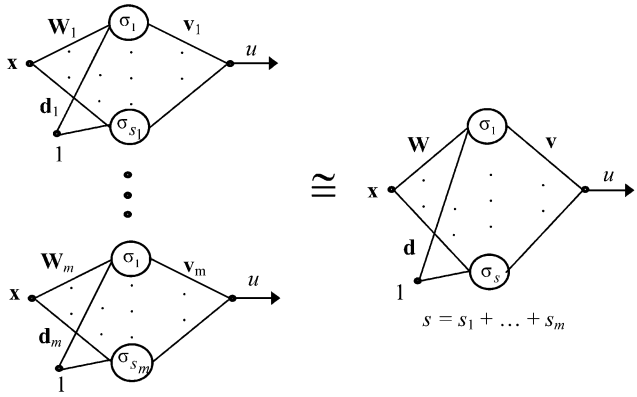

Fig. 3. Superposition of m s_g-node neural networks into one s-node network (summation symbols are omitted for simplicity).

The method is applied in Section V-B, where a neural network is trained to approximate the full aircraft trim map by superposition of several subnetworks. Generally speaking, the key to developing algebraic training techniques is to construct a matrix $\mathbf{S}$, through $\mathbf{N}$, that will display the desired characteristics. In the case of approximate input–output-based solutions, $\mathbf{S}$ must be of rank s whereas, s the number of nodes, is kept small to produce a parsimonious network.

D. Exact Matching of Function Gradient Data

Gradient-based training sets in the form $\{\mathbf{0} \,|\, \mathbf{a}^{k^T}]^T, 0, \mathbf{c}^k\}_{k=1,\ldots,p}$ are a special case for which the weight equations always exhibit an exact solution. These sets are referred to as gradient-based because knowledge of the function to be approximated mainly is provided by its gradients $\mathbf{c}^k$. At every training point, k, $\mathbf{c}^k$ is known for e of the neural network inputs, which are denoted by $\mathbf{x}$. The remaining $(q - e)$ inputs are denoted by $\mathbf{a}$. Input–output information also is available as $\mathbf{y}^k = [\mathbf{0} \,|\, \mathbf{a}^{k^T}]^T$ and $u^k = 0$, for any k. Hence, the output and gradient weight equations must be solved simultaneously. For convenience, the input-weight matrix is partitioned into weights corresponding to $\mathbf{x}, \mathbf{W}_\mathbf{x}$, and weights corresponding to $\mathbf{a}, \mathbf{W}_a$

$$\mathbf{W} = [\mathbf{W}_\mathbf{x} \,|\, \mathbf{W}_\mathbf{a}]. \tag{33}$$

Under the above conditions, the output weight equations (7) take the form

$$0 = \mathbf{v}^T \boldsymbol{\sigma}[\mathbf{W}_\mathbf{a} \mathbf{a}^k + \mathbf{d}] + b, \quad k = 1, \ldots, p \tag{34}$$

and are independent of the $\mathbf{W}_\mathbf{x}$ input weights because $\mathbf{x}^k$ equals zero in all p training triads. The gradient weight equations (12) depend on the $\mathbf{W}_\mathbf{a}$ input weights only implicitly

$$\mathbf{c}^k = \mathbf{W}_\mathbf{x}^T \{\mathbf{v} \otimes \boldsymbol{\sigma}'[\mathbf{W}_\mathbf{a} \mathbf{a}^k + \mathbf{d}]\}, \quad k = 1, \ldots, p \tag{35}$$

where (13) simplifies to

$$\mathbf{n}^k = \mathbf{W}_\mathbf{a} \mathbf{a}^k + \mathbf{d}, \quad k = 1, \ldots, p. \tag{36}$$

Equations (34)–(36) can be treated as three linear systems by assuming that all input-to-node values [n_i^k in (36)] are known.

The first linear system is obtained from (36), by reorganizing all n_i^k values into the following array:

$$\eta \equiv \left[\begin{array}{c} \mathbf{n}^1 \\ \hline \vdots \\ \hline \mathbf{n}^p \end{array}\right]. \tag{37}$$

When $s = p, \boldsymbol{\eta}$ becomes a known p^2-dimensional column vector. Then, the p^2 linear equations in (36) can be written in matrix notation as

$$\boldsymbol{\eta} = \mathbf{A}\mathbf{w}_\mathbf{a}. \tag{38}$$

$\mathbf{A}$ is a $ps \times (q-e+1)s$ matrix that is computed from all $\mathbf{a}^k$-input vectors in the training set. Each of these vectors contains $(q-e)$ elements, and the superscript indicates at which training pair each element has been evaluated

$$\mathbf{A} \equiv \left[\begin{array}{c|c|c} a_1^1\mathbf{I}_s & \cdots & a_{(q-e)}^1\mathbf{I}_s \;\; \mathbf{I}_s \\ \vdots & & \vdots \;\; \vdots \\ a_1^p\mathbf{I}_s & & a_{(q-e)}^p\mathbf{I}_s \;\; \mathbf{I}_s \end{array}\right]. \tag{39}$$

The only unknown parameters in (38) are the $\mathbf{a}$-input weights and the input bias. These are conveniently contained in the vector $\mathbf{w_a}$ that corresponds to the following rearrangement: $\mathbf{w_a} \equiv \text{Vec}[\mathbf{W_a} \,|\, \mathbf{d}]$.

Under the assumption of known n_i^k, the system in (34) becomes linear

$$\mathbf{b} = -\mathbf{S}\mathbf{v} \tag{40}$$

and it always can be solved for $\mathbf{v}$, provided $s = p$ and $\mathbf{S}$ is nonsingular. Subsequently, $\mathbf{v}$ can be treated as a constant, and (35) also becomes linear

$$\boldsymbol{\varsigma} = \mathbf{X}\mathbf{w}_\mathbf{x}. \tag{41}$$

In this system of equations, the unknowns consist of the $\mathbf{x}$-input weights that, for convenience, have been reorganized in the vector $\mathbf{w_x}, \mathbf{w_x} \equiv \text{Vec}(\mathbf{W_x})$. "Vec" indicates *Vec Operation*, which consists of columnwise reorganization of matrix elements into a vector [33]. The known gradients in the training set are assembled in the vector $\boldsymbol{\varsigma}$

$$\varsigma \equiv \left[\begin{array}{c} \mathbf{c}^1 \\ \hline \vdots \\ \hline \mathbf{c}^p \end{array}\right]. \tag{42}$$

$\mathbf{X}$ denotes a known $ep \times es$ sparse matrix composed of p block-diagonal sub-matrices of dimensions $e \times es$

$$\mathbf{X} \equiv \left[\begin{array}{ccc} \mathbf{B}^1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{B}^1 \\ \hline & \vdots & \\ \hline \mathbf{B}^p & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{B}^p \end{array}\right] \begin{array}{l} \left.\begin{array}{l}\\\\\end{array}\right\} e-\text{rows} \\ \vdots \\ \left.\begin{array}{l}\\\\\end{array}\right\} e-\text{rows} \end{array}. \tag{43}$$

The solution order of the above linear equations is key. The input-to-node values determine the nature of $\mathbf{S}$ and $\mathbf{X}$; repetitive values in $\boldsymbol{\eta}$ will render their determinants zero. The following algorithm determines an effective distribution for the elements in $\boldsymbol{\eta}$ so that the weight equations can be solved for the neural parameters in one step. Equation (38) is the first to be solved, since the input-to-node values are needed in the linear output and gradient weight [(40) and (41)]. $\mathbf{A}$ and $\boldsymbol{\varsigma}$ are determined from the training set, based on (39) and (42), choosing $s = p$. A strategy that produces a well-conditioned $\mathbf{S}$, with probability one, consists of generating $\boldsymbol{\eta}$ according to the rule

$$n_i^k = \begin{cases} r_i^k, & \text{if } i \neq k \\ 0, & \text{if } i = k \end{cases} \tag{44}$$

consistently with Section IV-A. Then, $\mathbf{w_a}$ is computed from (38) using the *left pseudoinverse* $\mathbf{A}^{\text{PI}}$

$$\hat{\mathbf{w}}_\mathbf{a} = \mathbf{A}^{\text{PI}}\boldsymbol{\eta} \tag{45}$$

$\hat{\mathbf{w}}_\mathbf{a}$ is the best approximation to the solution, as this overdetermined system is not likely to have a solution. When this value for $\mathbf{w_a}$ is substituted back in (38), an estimate to the chosen values (44) is obtained for $\boldsymbol{\eta}$

$$\hat{\boldsymbol{\eta}} = \mathbf{A}\hat{\mathbf{w}}_\mathbf{a}. \tag{46}$$

The elements of $\hat{\boldsymbol{\eta}}$ are used as input-to-node values in the output and gradient weight equations.

$\hat{\boldsymbol{\eta}}$ is computed on the basis of (40); therefore, the sigmoids are very nearly centered. While it is desirable for one sigmoid to be centered for a given input, $\mathbf{y}^k$, the same sigmoid should be close to saturation for any other known input in order to prevent ill-conditioning of $\mathbf{S}$. Considering that the sigmoids come close to being saturated for an input whose absolute value is greater than 5, it is found desirable for the input-to-node values in $\boldsymbol{\eta}$ to have variance of about 10. A factor f can be obtained for this purpose from the absolute value of the largest element in $\hat{\boldsymbol{\eta}}$; then the final values for $\boldsymbol{\eta}$ and $\mathbf{w_a}$ can be obtained by multiplying both sides of (46) by f

$$\begin{aligned} \boldsymbol{\eta} &= f\hat{\boldsymbol{\eta}} \\ \mathbf{w}_\mathbf{a} &= f\hat{\mathbf{w}}_\mathbf{a}. \end{aligned} \tag{47}$$

Subsequently, the matrix $\mathbf{S}$ can be computed from $\boldsymbol{\eta}$, and the system in (40) can be solved for $\mathbf{v}$. With the knowledge of $\mathbf{v}$ and $\boldsymbol{\eta}$, the matrix $\mathbf{X}$ can be formed as stated in (43), and the system (41) can be solved for $\mathbf{w_x}$. The matrices $\mathbf{S}$ and $\mathbf{X}$ in (40) and (41) are found to be consistently well-conditioned, rendering the solution of these linear systems straight-forward as well as highly accurate. Thus, both output and gradient weight equations, originally in the form of (34) and (35), are solved exactly for the network's parameters in a noniterative fashion. This algorithm is sketched in Fig. 4 and applied in Section V-C to train a gain-scheduled neural network controller.

E. Example: Neural Network Modeling of the Sine Function

A simple example is used to illustrate the algebraic solution approach. A sigmoidal neural network is trained to approximate the sine function $u = \sin(y)$ over the domain $0 \leq y \leq \pi$. The training set is comprised of the gradient and output information shown in Table I and takes the form $\{y^k, u^k, c^k\}_{k=1,2,3}$, with $q = e = 1$. As explained in the previous sections, the number

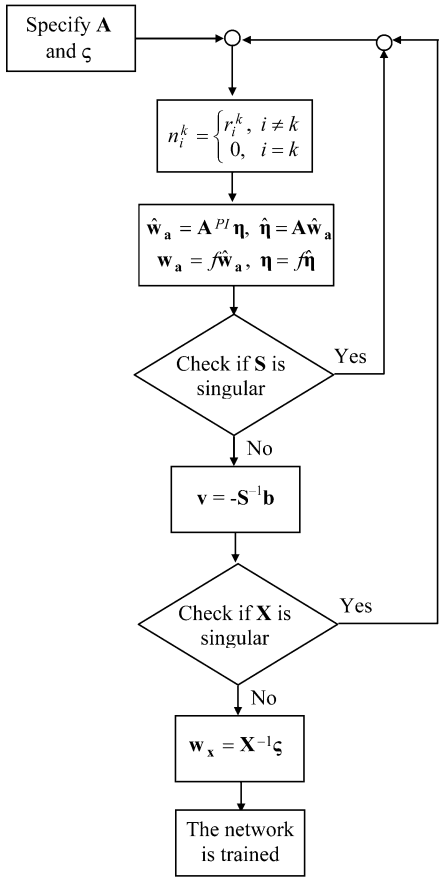

Fig. 4. Exact gradient-based algebraic algorithm.

of nodes and the values of the parameters of a sigmoidal neural network (Fig. 1) that matches this data exactly can be determined from the output weight (7)

$$u^1 = \mathbf{v}^T\boldsymbol{\sigma}[\mathbf{n}^1] + b = 0 \tag{48}$$
$$u^2 = \mathbf{v}^T\boldsymbol{\sigma}[\mathbf{n}^2] + b = 1 \tag{49}$$
$$u^3 = \mathbf{v}^T\boldsymbol{\sigma}[\mathbf{n}^3] + b = 0 \tag{50}$$

the gradient weight (12)

$$c^1 = \mathbf{w}^T\{\mathbf{v} \otimes \boldsymbol{\sigma}'[\mathbf{n}^1]\} = 1 \tag{51}$$
$$c^2 = \mathbf{w}^T\{\mathbf{v} \otimes \boldsymbol{\sigma}'[\mathbf{n}^2]\} = 0 \tag{52}$$
$$c^3 = \mathbf{w}^T\{\mathbf{v} \otimes \boldsymbol{\sigma}'[\mathbf{n}^3]\} = -1 \tag{53}$$

and the input-to-node weight (13)

$$\mathbf{n}^1 = \mathbf{W}\mathbf{y}^1 + \mathbf{d} = \mathbf{d} \tag{54}$$
$$\mathbf{n}^2 = \mathbf{W}\mathbf{y}^2 + \mathbf{d} = \mathbf{w}\pi/2 + \mathbf{d} \tag{55}$$
$$\mathbf{n}^3 = \mathbf{W}\mathbf{y}^3 + \mathbf{d} = \mathbf{w}\pi + \mathbf{d}. \tag{56}$$

The algebraic training approach computes a solution of the nonlinear weight equations by solving the linear systems obtained by separating the nonlinear sigmoid functions from their arguments (or input-to-node values). In this case, it is shown that the data (Table I) is matched exactly by a network with two nodes, i.e., with $\mathbf{W} = [w_{11}\ w_{12}]^T, \mathbf{d} = [d_1\ d_2]^T, \mathbf{n}^k = [n_1^k\ n_2^k]^T$, and $\mathbf{v} = [v_1\ v_2]^T$. Although there are 12 weights

TABLE I
DATA SET USED FOR ALGEBRAIC TRAINING OF A NEURAL NETWORK THAT MODELS THE SINE FUNCTION BETWEEN 0 AND π (IN RADIANS)

k:	y^k:	u^k:	c^k:
1	0	0	1
2	$\pi/2$	1	0
3	π	0	-1

(48)–(56) and only seven unknown parameters (**W, d, v**, and b), the input-to-node values ($\mathbf{n}^1, \mathbf{n}^2$ and $\mathbf{n}^3$) can be selected to make these overdetermined equations consistent, such that an exact solution for the parameters will exist. Suppose the input-to-node values $\mathbf{n}^1$ and $\mathbf{n}^3$ are chosen such that

$$\mathbf{n}^1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\mathbf{n}^3. \tag{57}$$

Then, if the following conditions are satisfied:

$$v_1 = v_2 \tag{58}$$
$$w_{11} = -w_{21} \tag{59}$$

where (48) becomes equivalent to (50), and (51) becomes equivalent to (53). From (54) and (56), it follows that the input weights must also satisfy the relationship:

$$\pi w_{11} = d_2 - d_1 = n_2^1 - n_1^1 \tag{60}$$

and, thus, from (55)

$$n_1^2 = n_2^2 = \frac{d_1 + d_2}{2} = \frac{n_1^1 + n_2^1}{2}. \tag{61}$$

With the assumptions in (58)–(59), (61) implies that the gradient (52) always holds. Therefore, the parameters v_1, w_{11}, and b can be determined from the remaining output and gradient equations (48), (49), and (51), which simplify to

$$v_1\left[\sigma\left(n_1^1\right) + \sigma\left(n_2^1\right)\right] + b = 0 \tag{62}$$
$$v_1\left[\sigma\left(n_1^2\right) + \sigma\left(n_2^2\right)\right] + b = 1 \tag{63}$$
$$v_1 w_{11}\left[\sigma'\left(n_1^1\right) + \sigma'\left(n_2^1\right)\right] = 1 \tag{64}$$

when subject to the above assumptions. Since the input bias $\mathbf{d}$ and the input-to-node values $\mathbf{n}^2$ and $\mathbf{n}^3$ all are specified in terms of $\mathbf{n}^1$ [as shown in (54), (57), and (61)], once $\mathbf{n}^1$ is chosen all network parameters can be determined by solving linear systems of equations. In this example, $\mathbf{n}^1$ is chosen to make the above weight equations consistent and to meet the assumptions in (57) and (60)–(61). It can be easily shown that this corresponds to computing the elements of $\mathbf{n}^1$ (n_1^1 and n_2^1) from the equation

$$\frac{\left(n_2^1 - n_1^1\right)}{\pi}\left[\sigma'\left(n_1^1\right) - \sigma'\left(n_2^1\right)\right] = 2\sigma\left(\frac{n_2^1 + n_1^1}{2}\right) - \sigma\left(n_1^1\right) - \sigma\left(n_2^1\right) \tag{65}$$

which is obtained by writing (62)–(64) solely in terms of $\mathbf{n}^1$, subject to (60)–(61).

Equation (65) is a nonlinear transcendental equation with two unknowns. However, by performing an appropriate change of variables it can be rewritten as two simultaneous algebraic equations with two unknowns (n_1^1 and n_2^1). The first algebraic equation is quadratic with respect to $\chi \equiv e^{n_2^1}$

$$A\chi^2 + B\chi + 1 = 0 \tag{66}$$

and is obtained from (65) through the change of variables shown in the Appendix. The constants A and B (introduced in the Appendix) are computed from a user-specified parameter K, which is defined by the second algebraic equation: $K = (n_2^1 - n_1^1)$. If K is chosen such that χ is real and positive, χ can be computed from (66) specifying both n_1^1 and n_2^1. Subsequently, all network weights are determined from the remaining algebraic equations derived above. The flowchart in Fig. 5 summarizes the sequence of solutions.

The output of a two-node sigmoidal network trained by this algebraic algorithm (Fig. 5) is shown in Fig. 6, for $K = 4$. In this graph, the output of the network, shown in a dotted line, is superimposed to the sine function (solid line) for comparison. The network is trained in one step, using a three-pair training set (Table I), which is matched exactly, and it achieves a MSE of $O(10^{-6})$ over a 50-point validation set. Similar results are obtained with other values of K. An equivalent performance can be obtained by training the two-node network with the LM algorithm [39] (e.g., the MATLAB LM training function), using 11 input–output training pairs and approximately 100 iterations (or epochs). The following sections demonstrate how the algebraic training approach can be used for approximating multidimensional functions by neural networks.

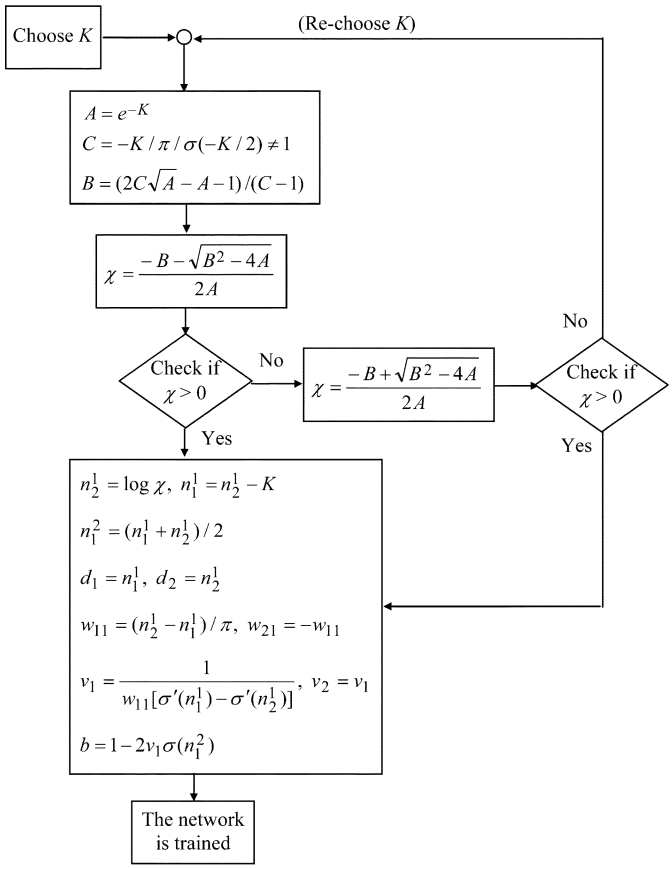

Fig. 5. Algebraic training algorithm for modeling the sine function by a two-node neural network.

IV. Neural Network Control of Aircraft by an Algebraic Training Approach

The algebraic algorithms derived in Section IV are demonstrated by training nonlinear neural networks that model forward and feedback controllers [23]. Control functions must be smooth, differentiable mappings of multidimensional nonlinear data. The dynamics of the aircraft to be controlled can be modeled by a nonlinear differential equation

$$\dot{\mathbf{x}}(t) = \mathbf{f}[\mathbf{x}(t), \mathbf{p}(t), \mathbf{u}(t)] \tag{67}$$

where the control function takes the general form

$$\mathbf{u}(t) = \mathbf{c}[\mathbf{y}_s(t), \mathbf{p}(t), \mathbf{y}_c(t)]. \tag{68}$$

The command input $\mathbf{y}_c$ can be viewed as some desirable combination of state and control elements. Plant motions and disturbances are sensed in the output vector $\mathbf{y}_s$

$$\mathbf{y}_s(t) = \mathbf{h}_s[\mathbf{x}(t), \mathbf{p}(t), \mathbf{u}(t)] \tag{69}$$

and $\mathbf{p}$ is a vector of plant and observation parameters. The full state vector of the aircraft $\mathbf{x} = [V\ \gamma q\ \theta r\ \beta p\ \mu]^T$ comprises airspeed V, path angle γ, pitch rate q, pitch angle θ, yaw rate r, sideslip angle β, roll rate p, and bank angle μ. The independent controls being generated are throttle δT, elevator δE, aileron δA, and rudder δR, i.e., $\mathbf{u} = [\delta T\ \delta S\ \delta A\ \delta R]^T$.

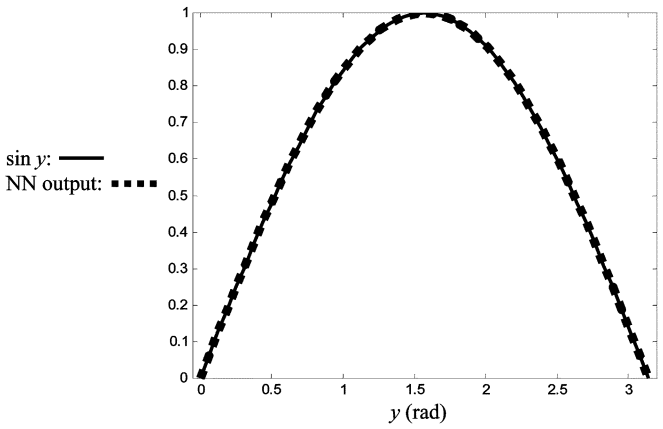

Fig. 6. Comparison between the output of the algebraically trained, two-node neural network (dotted line) and the sine function (solid line).

As shown in [23], the values of $\mathbf{u}$ at various equilibrium conditions are specified by the "trim" settings of the controls, and their gradients with respect to these flight conditions can be defined by the control gains of satisfactory linear controllers. Thus, both the functions and their derivatives are well-defined at an arbitrary number of operating points. The trim values and gradients can be specified as functions of velocity and altitude, which form a scheduling vector. The nonlinear controller is comprised of neural networks that express the trim control

settings and that provide feedback corrections that augment the aircraft stability and correct errors from the desired flight conditions $(\mathbf{y}_c)$.

A. Modeling of Longitudinal Trim Control Settings

In this section, the algorithms presented in Section IV-A and B are implemented to train a *forward neural network* that models the longitudinal trim map of an aircraft. The trim map comprises the equilibrium state of the controlled aircraft and the corresponding control settings [34]. The forward neural network provides the control commands required for equilibrium and is trained by matching input–output data exactly and gradient data approximately within a specified tolerance. With no disturbances or errors in the aircraft dynamic model, the control could be provided solely by the forward network.

Trim or equilibrium control settings $\mathbf{u}_c$ are defined for a given command input $\mathbf{y}_c$

$$\mathbf{u}_c = \mathbf{g}(\mathbf{x}_c, \mathbf{p}) = \mathbf{g}(\mathbf{y}_c) \tag{70}$$

such that

$$\mathbf{f}(\mathbf{x}_c, \mathbf{p}, \mathbf{u}_c) = \mathbf{f}[\mathbf{x}_c, \mathbf{p}, \mathbf{g}(\mathbf{y}_c)] = \mathbf{0} \tag{71}$$

with $\mathbf{x}_c$ computed from $\mathbf{y}_c$ and from the flight conditions $\mathbf{p}$. The aircraft trim map U_c is obtained by solving the steady-state equation (31) numerically p times over the aircraft's operational range OR

$$U_c(\mathbf{x}_c, \mathbf{p}) \equiv \left\{\mathbf{u}_c^k : \left(\mathbf{x}_c^k, \mathbf{p}^k\right) \in \text{OR} \right. \\ \left. \mathbf{f}\left(\mathbf{x}_c^k, \mathbf{p}^k, \mathbf{u}_c^k\right) = \mathbf{0}, k = 1, \ldots, p\right\}. \tag{72}$$

Local gradients of this hypersurface defined at each set point $(\mathbf{x}_c^k, \mathbf{u}_c^k)$ can be expressed as

$$\mathbf{C}_F^k \equiv \left(\left. \frac{\partial \mathbf{g}}{\partial \mathbf{y}_c} \right|_{\mathbf{y}_c^k} \right)^T, \quad k = 1, \ldots, p. \tag{73}$$

Here, a reduced order longitudinal-axis model is considered to illustrate the application of the algorithms derived in Section IV-A and B to problems that require precise matching of a relatively small set of data, i.e., $p \sim O(10^2)$. In Section V-B, the full aircraft model is used to illustrate the application of the training algorithm in Section IV-C for the synthesis of a larger data set $p \sim O(10^3)$, i.e., the full trim map. The longitudinal aircraft state vector contains velocity, path angle, pitch rate, and pitch angle: $\mathbf{x}_L = [V\ \gamma q\ \theta]^T$. The longitudinal controls are throttle position and elevator: $\mathbf{u}_L = [\delta T\ \delta E]^T$. The training of the longitudinal forward neural network is based on the trim data $\mathbf{y}_{c_L}^k, \mathbf{u}_{c_L}^k$, and $\mathbf{C}_{F_L}^k (k = 1, \ldots, p)$ that are consistent with the definitions of $\mathbf{x}_L$ and $\mathbf{u}_L$. The network's vector output $\mathbf{u}_{c_L}$ is given by the combination of two scalar, simply connected networks with velocity and path angle commands in $\mathbf{y}_{c_L} = [V_c\ \gamma_c]^T$

$$\mathbf{u}_{c_L} = \begin{bmatrix} \delta T_c \\ \delta E_c \end{bmatrix} \cong \begin{bmatrix} \mathrm{NN}_{F_1}\left(\mathbf{y}_{c_L}\right) \\ \mathrm{NN}_{F_2}\left(\mathbf{y}_{c_L}\right) \end{bmatrix}. \tag{74}$$

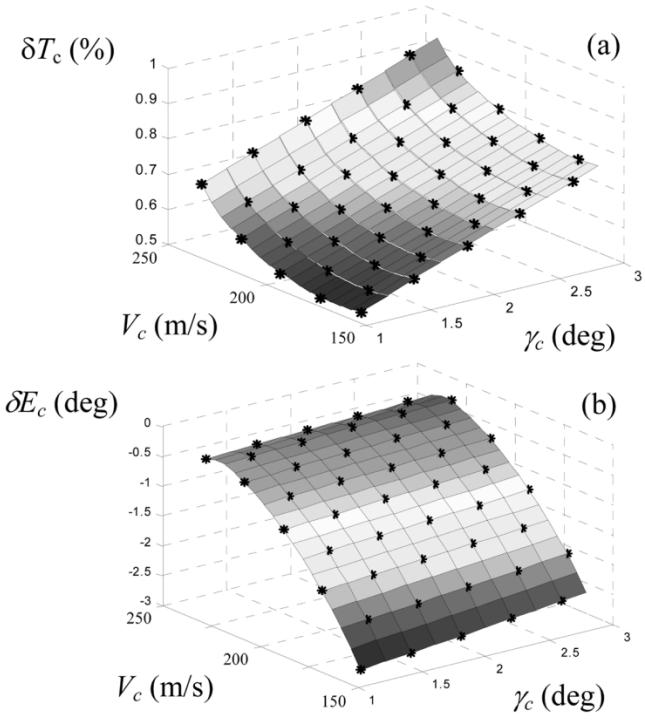

Fig. 7. Trim-map control surfaces, the asterisks symbolize corresponding training samples.

Every row of $\mathbf{C}_{F_L}^k$ provides the network gradient (1) for a control element. Fig. 7 shows the trim map being approximated; the intersections of the solid lines on the surfaces delineate the input space grid being plotted (the software interpolates between these points). The training set contains the trim data corresponding to 45 operating points describing different velocities and altitudes (also plotted in Fig. 7). Therefore, exact matching of the input–output data is obtained by a network with 45 nodes.

The parameters of NN_{F_1} and NN_{F_2} are determined from the corresponding weight equations(9) and (15) using the algorithm in Fig. 2. Initially, the parameters of NN_{F_1} obtained from the output equations produce a lumpy surface (Fig. 8), and the gradient tolerances are not immediately satisfied. The weights are further refined using the p-step gradient algorithm, finally producing the output surface in Fig. 9(a). For NN_{F_2}, the parameters that satisfy the desired tolerances [Fig. 9(b)] are obtained from the output weight equations alone (9) in only one step. The final neural output surfaces are plotted over a fine-grid input space in Fig. 9 to demonstrate the networks' interpolation abilities. The training time is a small fraction of a second on a contemporary desktop computer.

For comparison, a 45-node neural network is trained to approximate the same elevator input–output trim data [Fig. 7(b)] by means of the MATLAB 5.3 LM and RPROP training functions. Table II shows that the performance of the algebraic algorithm is superior to that of the two conventional algorithms in all respects. The output surface of the neural network trained by the LM algorithm $\mathrm{NN}_{\mathrm{LM}}$ is plotted in Fig. 10. The LM algorithm (which, in this case, outperforms RPROP) produces a network that has poor generalization properties when compared to the algebraically-trained network NN_{F_2} [Fig. 9(b)].

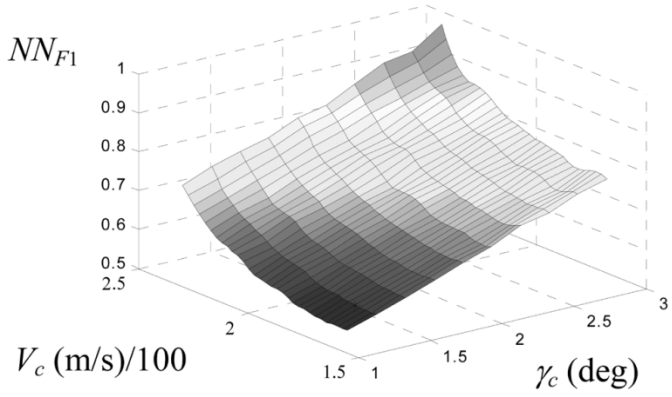

Fig. 8. Trim-throttle function approximation obtained from output weight equations alone.

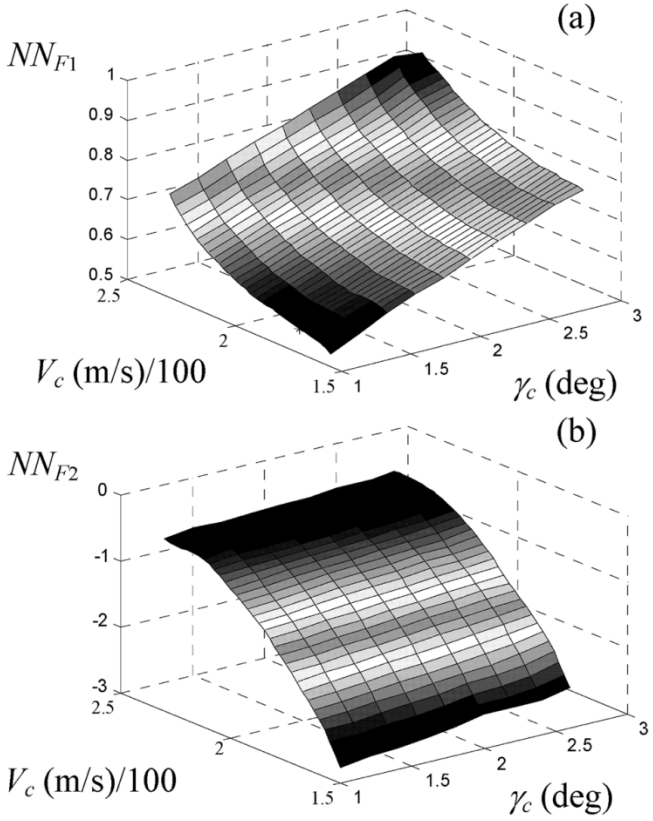

Fig. 9. Final trim-control function approximation where (a) is obtained from output and gradient weight equations and (b) is obtained from output weight equations.

With or without the use of derivative information, the algebraic approach minimizes data overfitting even when the network size is large because it addresses the input-to-node values and, hence, the level of saturation of the sigmoids, directly. In fact, using many nodes to approximate a smooth and relatively-flat surface [such as Fig. 9(b)] proves more challenging than approximating a highly nonlinear surface, because of the extra degrees of freedom.

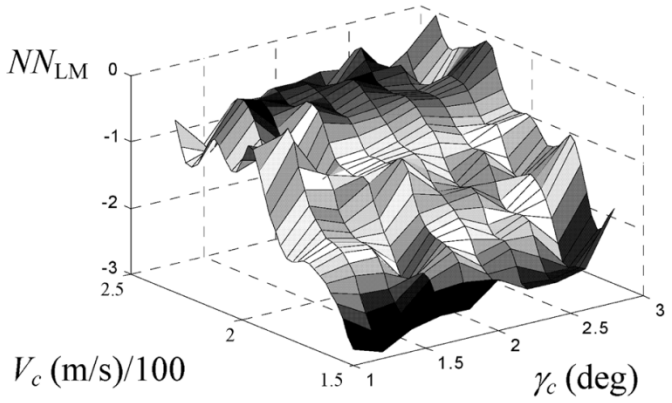

Fig. 10. Trim-elevator function approximation obtained by the LM algorithm.

B. Modeling of Coupled Longitudinal and Lateral Trim Control Settings

The previous section demonstrates how the algorithms in Section IV-A and B can be used to model a training set with smooth data that needs to be matched closely. In some applications, the number of training points is much larger, and a parsimonious network that synthesizes the data with fewer nodes is preferred. The approach presented in Section IV-C can be used to train such a network with lesser computational complexity than conventional optimization algorithms. As an example, the full aircraft trim map (72) is modeled by a forward neural network $\mathbf{NN}_F$ (Fig. 11) that computes both longitudinal and lateral trim control settings $\mathbf{u}_c = [\delta T_c\ \delta S_c\ \delta A_c\ \delta R_c]^T$, given the command input $\mathbf{y}_c = [V_c\ \gamma_c\ \mu_c\ \beta_c]^T$ and the desired altitude H_c

$$\begin{bmatrix} \mathbf{u}_c \\ \theta_c \\ \dot{\psi}_c \end{bmatrix} \cong \mathbf{NN}_F(\mathbf{y}_c, H_c). \tag{75}$$

To every value of $\mathbf{u}_c$, there corresponds a unique pair of pitch angle θ_c and yaw rate $\dot{\psi}_c$ that, together with $\mathbf{u}_c$, specify the steady maneuver (71) commanded by $\mathbf{y}_c$. Therefore, the functional relationship between these two parameters and the command input also is modeled by the forward neural network. A sampled description of the full trim map U_c (72) is obtained by solving (71) numerically throughout the full operating range OR using a least-squares algorithm [24]. For the aircraft, OR is defined as the set of all possible steady maneuvers involving some combination of airspeed, altitude, path angle, bank angle, and sideslip, i.e., OR$\equiv \{V, H, \gamma, \mu, \beta\}$. The aircraft physical characteristics and specifications suggest the following limits for these state variables:

$$\text{OR}: \begin{cases} 75\ \text{m/s} & \le V \le & 250\ \text{m/s} \\ 0\ \text{m} & \le H \le & 15\,000\ \text{m} \\ -6\,\text{deg} & \le \gamma \le & 6\,\text{deg} \\ -21\,\text{deg} & \le \mu \le & 21\,\text{deg} \\ -5\,\text{deg} & \le \beta \le & 5\,\text{deg} \end{cases}. \tag{76}$$

The actual boundaries of the multidimensional envelope OR are found while solving (71), sampling the ranges in (76) with the following intervals: $\Delta V = 5$ m/s, $\Delta H = 1\,000$ m, $\Delta\gamma = 1^\circ$, $\Delta\mu \sim 5^\circ$, and $\Delta\beta = 1^\circ$ [24].

TABLE II
PERFORMANCE COMPARISON OF TRAINING ALGORITHMS FOR THE APPROXIMATION OF A SCALAR FUNCTION BY A 45-NODE SIGMOIDAL NEURAL NETWORK

Algorithm:	Time (Scaled):	Flops:	Lines of code (MATLAB®):	Epochs:	Final error:
Algebraic	1	2×10^5	8	1	0
Levenberg-Marquardt	50	5×10^7	150	6	10^{-26}
Resilient Backprop.	150	1×10^7	100	150	0.006

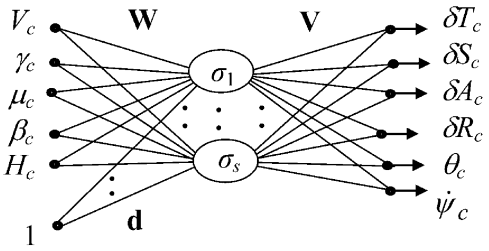

Fig. 11. Forward neural network architecture (summation symbols are omitted for simplicity).

A representative training set $\{\mathbf{y}_c^k, [\mathbf{u}_c^{k^T} \theta_c^k \ \dot{\psi}_c^k]^T\}_{k=1,\ldots,2\,696}$, is obtained from the following combinations:

$$\{\mu, \beta\} = \left\{ \begin{bmatrix} -4 & -1 & 1 & 4 \\ -5 & -2 & 3 & 5 \\ -5 & -3 & 2 & 5 \\ -4 & 0 & 0 & 4 \\ -5 & -1 & 1 & 5 \end{bmatrix} \right. \times \left. \begin{bmatrix} -21 & -14 & -21 & -21 \\ -15 & -6 & -1 & -15 \\ 0 & 4 & 9 & 0 \\ 14 & 0 & 15 & 8 \\ 20 & 21 & 19 & 20 \end{bmatrix} \right\} \text{(in degress)} \quad (77)$$

and randomly choosing values of V, H and γ in increments of $\Delta V_{\text{train}} = 30$ m/s, $\Delta H_{\text{train}} = 2\,000$ m, and $\Delta\gamma_{\text{train}} \sim 10^\circ$, respectively. Two additional sets of data are used for validation purposes: one with 39764 operating points and one with 2629 operating points. They are obtained from the sampled description of U_c by considering the (μ, β) combinations excluded in (77) and randomly selected values of V, H, and γ. In this case, gradient information is omitted for simplicity. Because there are 2696 training pairs, it is convenient to seek an approximate matching of the data, synthesizing the input–output information by a parsimonious network with $s < p$.

Conventional optimization-based techniques can successfully train small networks, with $s < p$, provided $p \sim O(100)$, but they quickly run out of memory if a large set is used at once (i.e., by *batch training*). A common approach is to divide the set into many subsets that are used to train the network *sequentially*. This procedure can be particularly arduous for conventional algorithms, as the network is likely to forget previously learned subsets while it is being trained with new ones. Instead, LM batch training is implemented here to train m subnetworks that are then combined into a single network in one step by the algebraic algorithm of Section IV-C. Twenty training subsets are obtained from the twenty (μ, β) combinations in (77). Each subset $\{\mathbf{y}_c^k, [\mathbf{u}_c^{k^T} \theta_c^k \ \dot{\psi}_c^k]^T\}_{k=p_{g-1}+1,\ldots,p_g}$ contains the trim data corresponding to approximately 135 equilibria and can be modeled by a network of the type shown in Fig. 11 with parameters $\mathbf{W}_g, \mathbf{d}_g$, and $\mathbf{V}_g$ $(b_g = 0)$, where $g = 1, \ldots, 20$. It is easily found that each subset can be approximated by a ten-node network with a MSE of $O(10^{-6})$ and excellent generalization properties. The MATLAB LM training function is implemented for this purpose.

Subsequently, according to the algorithm in Section IV-C, a network that models the full training set (the collection of all twenty subsets) can be obtained algebraically from the former subnetworks. For this full forward neural network $\mathbf{NN}_F s = 200$, since $s_g = 10$ for $\forall g$ and $m = 20$, and the input weights $\mathbf{W}$ and $\mathbf{d}$ are obtained from (29)–(30). The output weights are computed from the vector-output equivalent of (32)

$$\mathbf{V} = (\mathbf{S}^{\text{PI}}\mathbf{U})^T \quad (78)$$

similarly to the process illustrated in Fig. 3. The matrix $\mathbf{S}$ is computed from (10), based on the values in (31), and $\mathbf{U}$ contains all of the output training data

$$\mathbf{U} = \begin{bmatrix} \mathbf{u}_c^1 & \cdots & \mathbf{u}_c^p \\ \theta_c^1 & \cdots & \theta_c^p \\ \dot{\psi}_c^1 & \cdots & \dot{\psi}_c^p \end{bmatrix}^T. \quad (79)$$

The generalization capabilities of the full network are tested throughout OR by computing the MSE between the trim settings at the validation points to those computed by $\mathbf{NN}_F$ and by plotting the projection of the neural mapping onto three-dimensional (3-D) space. The MSE is found to be approximately 3×10^{-5} rad or rad/s for both of the validation sets described above. A representative surface approximated by $\mathbf{NN}_F$ is plotted in Fig. 12 and compared to trim data from the first validation set (Fig. 13) by holding γ_c, μ_c, and β_c constant and computing the output over a fine-grid V_c-H_c input space. These results are typical among all graphical comparisons performed elsewhere in OR. Hence, it is verified that good generalization properties

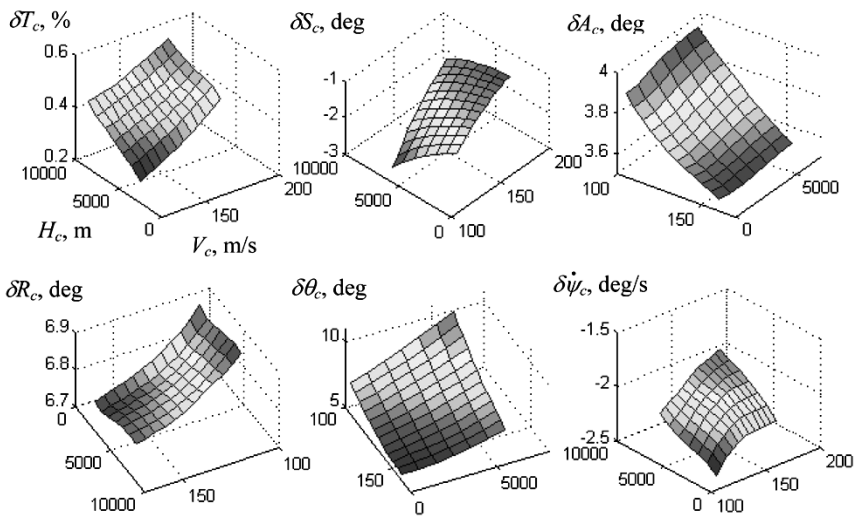

Fig. 12. Trim control surfaces as modeled by the forward neural network over a $\{V_c, H_c\}$-input space, with remaining inputs fixed at $(\gamma_c, \mu_c, \beta_c) = (3^\circ, 14^\circ, 4^\circ)$.

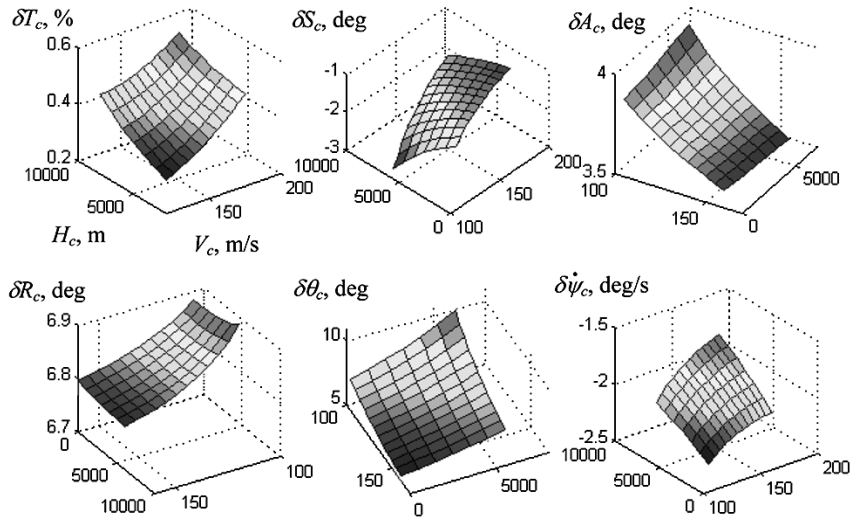

Fig. 13. Actual trim control surfaces plotted over a $\{V_c, H_c\}$-input space, with remaining inputs fixed at $(\gamma_c, \mu_c, \beta_c) = (3^\circ, 14^\circ, 4^\circ)$.

are obtained consistently across the full operating domain, indicating that overfitting does not occur.

C. Nonlinear Feedback Neural Networks

Feedback neural networks that interpolate linear control matrices obtained at selected operating points have been proposed in several applications (e.g., [35] and [36]). The algorithm derived in Section IV-D can be used to obtain these nonlinear neural networks algebraically, by solving systems of linear equations. Here, the method is demonstrated with a feedback neural network that controls the longitudinal aircraft dynamics throughout the steady-level flight envelope, as in [23].

Control laws that satisfy desired engineering criteria can be designed for chosen operating points to provide a set of locally-optimal gains $\{\mathbf{C}_B\}_k$. Typically, a controller is obtained by interpolating these local designs to intermediate operating regions by means of the scheduling vector $\mathbf{a}$, introduced above, through a procedure referred to as gain scheduling. Here, a nonlinear feedback controller is devised by using the local gain matrices to train a neural network that, at any given time, computes the deviation from the nominal controller given the state deviation and the flight condition.

The training set is found by inspection, from the control law. For every kth operating point the neural network gradient is given by the control gains at that point

$$\left.\frac{\partial \mathbf{u}[\mathbf{x}(t)]}{\partial \mathbf{x}(t)}\right|_{\mathbf{x}_0, \mathbf{a}^k} = -\mathbf{C}_B^k. \tag{80}$$

$\mathbf{C}_B^k$ and $\mathbf{a}^k$ are the optimal gain matrix and the scheduling vector evaluated at the kth operating condition; every row of $\mathbf{C}_B^k$ provides the gradient (1) for a control element. Also, the following input–output condition always holds:

$$\Delta\mathbf{u}[\Delta\mathbf{x}(t), \mathbf{a}(t)]|_{\Delta\mathbf{x}_0, \mathbf{a}^k} = \mathbf{0} \tag{81}$$

producing a training set of the form considered in Section IV-D. The longitudinal feedback neural network $\mathbf{NN}_{B_L}$ is composed

TABLE III
COMPARISON OF NEURAL NETWORK GRADIENTS WITH ACTUAL FEEDBACK GAINS AT THREE VALIDATION POINTS

Validation Point:	$\mathbf{a}$ = [210 (m/s), 15 (Km)]^T	$\mathbf{a}$ = [165 (m/s), 8 (Km)]^T	$\mathbf{a}$ = [135 (m/s), 4 (Km)]^T
$\frac{\partial \mathbf{NN}_{B_L}}{\partial \mathbf{x}_L}$:	$\begin{bmatrix} 0.945 & 4.006 & -0.593 & -2.600 \\ 0.006 & -1.424 & -0.193 & -0.093 \end{bmatrix}$	$\begin{bmatrix} 0.502 & 0.785 & -0.256 & -1.118 \\ 0.003 & -0.918 & -0.120 & 0.049 \end{bmatrix}$	$\begin{bmatrix} 0.367 & 0.311 & -0.176 & -0.753 \\ 0.002 & -0.737 & -0.102 & 0.059 \end{bmatrix}$
$\mathbf{C}_{B_L}$:	$\begin{bmatrix} 0.947 & 4.024 & -0.594 & -2.598 \\ 0.006 & -1.392 & -0.190 & -0.086 \end{bmatrix}$	$\begin{bmatrix} 0.506 & 0.816 & -0.261 & -1.129 \\ 0.003 & -0.897 & -0.118 & 0.053 \end{bmatrix}$	$\begin{bmatrix} 0.374 & 0.465 & -0.208 & -0.796 \\ 0.004 & -1.091 & -0.183 & -0.197 \end{bmatrix}$

of two scalar networks with the architecture shown in Fig. 1, one for each control element

$$\Delta \mathbf{u}_L(t) = \begin{bmatrix} \Delta\delta T(t) \\ \Delta\delta E(t) \end{bmatrix} = \begin{bmatrix} \mathrm{NN}_{B_1}(\Delta \mathbf{x}_L(t), \mathbf{a}) \\ \mathrm{NN}_{B_2}(\Delta \mathbf{x}_L(t), \mathbf{a}) \end{bmatrix}. \quad (82)$$

The neural networks' size and parameters are determined from the exact gradient-based solution (Section IV-D) in one step.

Suppose the feedback gain matrices have been designed at 34 operating points also referred to as *design points*. Then, two 34-node sigmoidal networks can match these gains exactly and generalize them throughout the steady-level flight envelope [23]. The network weights $\mathbf{W}, \mathbf{d}, \mathbf{V}$, and b are computed from (38), (40), and (41), according to the algorithm in Fig. 4. The algebraic training algorithm executes in about 1 s for each network. Because the weight equations are solved exactly, the error at the 34 initialization points (Fig. 7) is identically zero. The generalization properties are tested by producing a validation set of feedback gains at nontraining points and comparing them to the network gradients computed from the trained weights, based on (5). The validation set is obtained by designing the gain matrices at 290 operating points within the flight envelope. The L^2 norm of the error between this gradient and the corresponding gains equals 0.045 on average and has a maximum value of 0.14. A comparison of network gradients $(\partial \mathbf{NN}_{B_L}/\partial \mathbf{x}_L)$ and actual gains $(\mathbf{C}_{B_L})$ is shown in Table III for sample interpolation points chosen from the validation set. This accuracy translates into excellent control performance everywhere in the flight envelope, as shown in [23] and [24]. While gradient-based training constitutes an added degree of complexity for conventional optimization-based algorithms, it is handled just as easily as input–output-based training by the algebraic approach.

V. CONCLUSION

An algebraic training approach that affords a great deal of insight into neural approximation properties and applications is developed. The underlying principles are illustrated for the batch training of a classical feedforward sigmoid architecture. The techniques developed in this paper match input–output and gradient information approximately or exactly by neural networks. The adjustable parameters or weights are determined by solving linear systems of equations. Four algebraic algorithms are derived based on the exact or approximate solution of input–output and gradient weight equations. Their effectiveness is demonstrated by training forward neural networks, which synthesize a transport aircraft's trim map, and feedback neural networks, which produce a gain-scheduled control design. All implementations show that algebraic neural network training is fast and straightforward for virtually any noise-free nonlinear function approximation, and it preserves the networks' generalization and interpolation capabilities.

APPENDIX

In (65), let $(n_2^1 - n_1^1) = K$, where K is a user-specified constant. Then, n_1^1 can be written in terms of n_2^1, as $n_1^1 = n_2^1 - K$, which, when substituted back in (65), leads to

$$\frac{K}{\pi}\left[\sigma'\left(n_2^1 - K\right) - \sigma'\left(n_2^1\right)\right] = 2\sigma\left(n_2^1 - K/2\right) - \sigma\left(n_2^1 - K\right) - \sigma\left(n_2^1\right). \quad (82)$$

For the chosen sigmoid function, $\sigma(n) = (e^n - 1)/(e^n + 1)$ and $\sigma'(n) = 2e^n/(e^n + 1)^2$. Therefore, by renaming the quantities $\chi \equiv e^{n_2^1}$ and $A \equiv e^{-K}$, (82) can be written as an algebraic equation with respect to χ

$$\frac{2K}{\pi}\left[\frac{A\chi}{(A\chi+1)^2} - \frac{\chi}{(\chi+1)^2}\right] = \frac{2(\sqrt{A}\chi - 1)}{(\sqrt{A}\chi + 1)} - \frac{(A\chi - 1)}{(A\chi + 1)} - \frac{(\chi - 1)}{(\chi + 1)} \quad (83)$$

and, with some manipulation, it can be simplified to

$$\frac{2K}{\pi}\frac{(1+\sqrt{A})}{(1-\sqrt{A})}(\sqrt{A}\chi + 1)^2 = (A\chi + 1)(\chi + 1). \quad (84)$$

The terms in the above equation can be rearranged to obtain a quadratic equation (65), where, for convenience, the following constants are introduced:

$$B \equiv \frac{2\sqrt{A}C - A - 1}{C - 1} \quad (85)$$

with $C \neq 1$, and

$$C \equiv \frac{-K}{\pi\sigma(-K/2)}. \quad (86)$$

REFERENCES

[1] *Handbook of Intelligent Control*, D. A. White and D. A. Sofge, Eds., Van Nostrand, New York, 1992, pp. 65–86. P. J. Werbos, Neurocontrol and Supervised Learning: An Overview and Evaluation.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning*. Cambridge, MA: MIT Press, 1998.

[3] S. Lane and R. F. Stengel, "Flight control design using nonlinear inverse dynamics," *Automatica*, vol. 24, no. 4, pp. 471–483, 1988.
[4] M. G. Cox, "Practical spline approximation," in *Lecture Notes in Mathematics 965: Topics in Numerical Analysis*, P. R. Turner, Ed. New York: Springer-Verlag, 1982.
[5] A. Antoniadis and D. T. Pham, "Wavelets and statistics," in *Lecture Notes in Statistics 103*. New York: Springer-Verlag, 1995.
[6] C. K. Chui, *An Introduction to Wavelets*. New York: Academic, 1992.
[7] T. Lyche, K. Mørken, and E. Quak, "Theory and algorithms for nonuniform spline wavelets," in *Multivariate Approximation and Applications*, N. Dyn, D. Leviatan, D. Levin, and A. Pinkus, Eds, Cambridge, U.K.: Cambridge Univ. Press, 2001.
[8] J. H. Friedman, "Multivariate adaptive regression splines," *Ann. Statist.*, vol. 19, pp. 1–141, 1991.
[9] S. Karlin, C. Micchelli, and Y. Rinott, "Multivariate splines: A probabilistic perspective," *J. Multivariate Anal.*, vol. 20, pp. 69–90, 1986.
[10] C. J. Stone, "The use of polynomial splines and their tensor products in multivariate function estimation," *Ann. Statist.*, vol. 22, pp. 118–184, 1994.
[11] K. S. Narendra and K. Parthasaranthy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Netw.*, vol. 1, no. 1, pp. 4–27, Jan. 1990.
[12] K. J. Hunt, D. Sbarbaro, R. Zbikowski, and P. J. Gawthrop, "Neural networks for control systems—A survey," *Automatica*, vol. 28, no. 6, pp. 1083–1112, 1992.
[13] D. Linse and R. F. Stengel, "Identification of aerodynamic coefficients using computational neural networks," *J. Guid. Control Dyn.*, vol. 16, no. 6, pp. 1018–1025, 1993.
[14] H. Zhao, O. J. Hao, T. J. McAvoy, and C. H. Chang, "Modeling nutrient dynamics in sequencing batch reactors," *J. Environ. Eng.*, vol. 123, pp. 311–319, 1997.
[15] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Trans. Neural Netw.*, vol. 9, no. 5, pp. 987–995, Sep. 1998.
[16] D. E. Rumelhart, G. E. Inton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.
[17] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Netw.*, vol. 1, no. 4, pp. 295–308, 1988.
[18] A. K. Rigler, J. M. Irvine, and T. P. Vogl, "Rescaling of variables in back-propagation learning," *Neural Netw.*, vol. 3, no. 5, pp. 561–573, 1990.
[19] A. N. Kolmogorov, "On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition," *Dokl. Akad. Nauk SSSR*, vol. 114, pp. 953–956, 1957.
[20] G. Cybenko, "Approximation by superposition of a sigmoidal function," *Math. Control. Signals, Syst.*, vol. 2, pp. 303–314, 1989.
[21] K. Hornik, M. Stinchcombe, and H. White, "Multi-layer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, pp. 359–366, 1989.
[22] A. R. Barron, "Universal approximation bounds for superposition of a sigmoidal function," *IEEE Trans. Inf. Theory*, vol. 39, no. 3, pp. 930–945, May 1993.
[23] S. Ferrari and R. F. Stengel, "Classical/neural synthesis of nonlinear control systems," *J. Guid. Control Dyn.*, vol. 25, no. 3, pp. 442–448, 2002.
[24] S. Ferrari, "Algebraic and adaptive learning in neural control systems," Ph.D. dissertation, Princeton Univ., Princeton, NJ, 2002.
[25] S. Ferrari, "Algebraic and adaptive learning in neural control systems," Ph.D. dissertation, Princeton Univ., Princeton, NJ, 2002. Forward neural network, Sec. 4.4.
[26] S. Ferrari and R. F. Stengel, "Algebraic training of a neural network," in *Proc. Amer. Control Conf.*, Arlington, VA, Jun. 2001.
[27] S. Ferrari, "Algebraic and adaptive learning in neural control systems," Ph.D. dissertation, Princeton Univ., Princeton, NJ, 2002. Algebraically-Constrained Adaptive Critic Architecture, Sec. 5.3.
[28] G. Strang, *Linear Algebra and Its Applications*, 3rd ed. Orlando, FL: Harcourt Brace Janovich, 1988.
[29] D. Nguyen and B. Widrow, "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights," in *Proc. Int. Joint Conf. Neural Networks*, vol. III, San Diego, CA, 1990, pp. 21–26.
[30] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct. 1990.
[31] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE Trans. Neural Netw.*, vol. 5, no. 6, pp. 989–993, Nov. 1994.
[32] M. Reidmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *Proc. IEEE Int. Conf. NN (ICNN)*, San Francisco, CA, 1993, pp. 586–591.
[33] A. Graham, *Kronecker Products and Matrix Calculus: With Applications*, Chichester, U.K.: Ellis Horwood, 1981.
[34] L. S. Cicolani, B. Sridhar, and G. Meyer, "Configuration management and automatic control of an augmentor wing aircraft with vectored thrust,", NASA Tech. Paper, TP-1222, 1979.
[35] M. A. Sartori and P. J. Antsaklis, "Implementations of learning control systems using neural networks," *IEEE Control Sys. Mag.*, vol. 12, no. 2, pp. 49–57, 1992.
[36] J. Neidhoefer and K. Krishnakumar, "Neuro-gain approximation (a continuous approximation to the nonlinear mapping between linear controllers)," *Intell. Eng. Syst. Through Artif. Neural Netw.*, vol. 6, pp. 543–550, 1996.

Silvia Ferrari (M'04) received the B.S. degree from Embry-Riddle Aeronautical University, Daytona Beach, FL, and the M.A. and Ph.D. degrees from Princeton University, Princeton, NJ.

She is currently an Assistant Professor of mechanical engineering and materials science at Duke University, Durham, NC, where she directs the Laboratory for Intelligent Systems and Controls (LISC). Her principal research interests are robust adaptive control of aircraft, learning and approximate dynamic programming, and distributed sensor planning.

Dr. Ferrari is a Member of the American Institute of Aeronautics and Astronautics. She received the ONR Young Investigator Award in 2004, the Wallace Memorial Honorific Fellowship in Engineering in 2002, the Zonta International Amelia Earhart Fellowship Award in 2000 and 2001, the AAS Donald K. Deke Slayton Memorial Scholarship in 2001, the ASME Graduate Teaching Fellowship in 2001, and the AIAA Guidance, Navigation, and Control Graduate Award in 1999.

Robert F. Stengel (M'77–SM'83–F'93) received the S.B. degree from the Massachusetts Institute of Technology, Cambridge, in 1960 and the M.S.E., M.A., and Ph.D. degrees from Princeton University, Princeton, NJ, in 1960, 1965, 1966, and 1968, respectively.

He is currently a Professor and former Associate Dean of Engineering and Applied Science at Princeton University, where he directs the undergraduate program in robotics and intelligent systems. He has taught courses on robotics and intelligent systems, control and estimation, aircraft flight dynamics, and space flight. Prior to his 1977 Princeton appointment, he was with The Analytic Sciences Corporation, Charles Stark Draper Laboratory, U.S. Air Force, and the National Aeronautics and Space Administration. A principal designer of the Project Apollo Lunar Module manual attitude control logic, he also contributed to the design of the space shuttle guidance and control system. From 1977 to 1983, he was Director of Princeton's Flight Research Laboratory, where he investigated aircraft flying qualities, digital control, and system identification using two fly-by-wire aircraft, and Vice Chairman of the Congressional Aeronautical Advisory Committee. He wrote the books, *Optimal Control and Estimation* (New York: Dover Publications, 1994) and *Flight Dynamics* (Princeton, NJ: Princeton University Press, 2004), and he has authored or coauthored numerous technical papers and reports. Current research interests include systems biology, nonlinear, robust, and adaptive control systems, and optimization.

Dr. Stengel is a Fellow of the American Institute of Aeronautics and Astronautics (AIAA). He received the American Automatic Control Council (AACC) John R. Ragazzini Control Education Award in 2002, the AIAA Mechanics and Control of Flight Award in 2000, and the FAA's first annual Excellence in Aviation Award in 1997. He was Associate Editor at Large of the IEEE TRANSACTIONS ON AUTOMATIC CONTROL.