Proceedings of the American Control Conference
Arlington, VA June 25-27, 2001

Algebraic Training of a Neural Network

Silvia Ferrari* and Robert F. Stengel†

Princeton University
Department of Mechanical and Aerospace Engineering
Princeton, NJ 08544

Abstract

A novel algebraic neural network training technique is developed and demonstrated on two well-known architectures. This approach suggests an innovative, unified framework for analyzing neural approximation properties and for training neural networks in a much simplified way. Various implementations show that this approach presents numerous practical advantages; it provides a trouble-free non-iterative systematic procedure to integrate neural networks in control architectures, and it affords deep insight into neural nonlinear control system design.

1. Introduction

Computational neural networks are massively parallel computational paradigms inspired by biological neural formations. They are used in a variety of applications because they can learn by example and provide excellent universal function approximation for multivariate input/output spaces. Particularly, they afford a general approach for modeling, identification, and control of nonlinear systems that shows great promise, as neural networks can potentially adjust to complex situations online thanks to their brain-like generalizing and adaptive capabilities.

Considerable effort has gone into the mathematical investigation of networks' approximation properties [1-3]. Whereas these results appear attractive, they provide little insight into practical, key questions such as, "What architecture should be used", and "How many nodes are required in each layer"? This paper describes a novel training technique that provides a general framework for answering these questions as well as for incorporating available control theory that could later be used "intelligently" [4].

The typical search criterion, used in virtually all supervised learning algorithms, consists of minimizing some measure of the error between the desired input/output (and/or derivative) information and the actual network's performance. The approach taken here consists of formulating training as a root-finding problem whose solution achieves exact fitting of the training set. Although related in principle, the problems of minimization and multidimensional root finding are substantially different in practice. The problem of minimizing some form of neural network error appears computationally more tractable, but it may not solve the problem of exact fitting. On the other hand, solving the corresponding nonlinear equations, here referred to as *initialization equations*, appears virtually impossible for any decent-sized network. So what is the reason behind attempting to solve a harder version of the same problem? As it happens, the initialization equations can easily be transformed into linear equations that bring about a much simplified training methodology and afford deep insight into neural approximators.

The method is demonstrated by training a *forward neural network* that models a transport aircraft's trim map. The approach also proved highly effective and insightful in training neural networks to approximate gain-scheduled controllers, as described in [4]. The implementations show that, in addition to being relevant from a theoretical point of view, this methodology presents numerous practical advantages inherent to the control field.

2. Input/Output Algebraic Training of a Simply Connected Neural Network

Let $h:\Re^q \to \Re$ denote a smooth scalar function that is to be approximated, where q is the dimension of its input. Suppose the function is not analytically known, but a set of input/output samples $\{\mathbf{y}^k, u^k\}_{k=1,\ldots,p}$ can be generated such that, at any k, $u^k = h(\mathbf{y}^k)$. Then, this set of samples constitutes what is typically referred to as a network input/output training set. A sample, simply connected, scalar-output network is shown in Fig. 1.

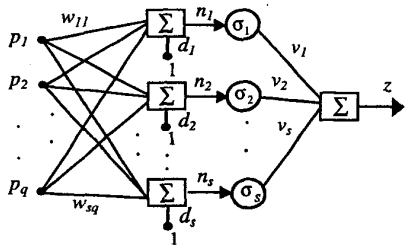

Figure 1. Sample scalar-output network with q-inputs and s-nodes in the hidden layer.

The same nonlinear function is used in all nodes. Later sections demonstrate how the same principles are applicable to even more general architectures.

---

* Graduate Student.
† Professor. Fellow IEEE, AIAA.
Presented at the 2001 American Control Conference, Arlington, VA, June 2001.

The output of the network is computed as the nonlinear transformation of the weighted sum of the input, **p**, and a bias **d**:

$$z = \mathbf{v}^T \boldsymbol{\sigma}[\mathbf{W}\mathbf{p} + \mathbf{d}] \tag{1}$$

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

$$\boldsymbol{\sigma}[\mathbf{n}] \equiv [\sigma(n_1) \ \cdots \ \sigma(n_s)]^T \tag{2}$$

where:

$$\mathbf{n} = \mathbf{W}\mathbf{p} + \mathbf{d} \tag{3}$$

W and **v** contain the input and output weights, respectively. Together with the input bias, **d**, they constitute the adjustable parameters of the network. The order of differentiability of eq. 1 is the same as that of the activation function, $\sigma(\bullet)$. Given that sigmoid functions are infinitely differentiable, the derivative of the network output with respect to its inputs is:

$$\frac{\partial z}{\partial p_j} = \sum_{i=1}^{s} \frac{\partial z}{\partial n_i} \frac{\partial n_i}{\partial p_j} = \sum_{i=1}^{s} v_i \sigma'(n_i) w_{ij}, \quad j = 1, \ldots, q \tag{4}$$

$\sigma'(\bullet)$ denotes the derivative of the sigmoidal function with respect to its scalar input. w_{ij} denotes the element in the i^{th}-row and the j^{th}-column of the matrix **W**, and it represents the interconnection weight between the j^{th}-input and the i^{th}-node of the network.

The computational neural network achieves exact fitting of the training set when, given the input $\mathbf{y}^k$, it produces the corresponding output u^k i.e., $z(\mathbf{y}^k) = u^k$, for any k. This is equivalent to stating that the neural adjustable parameters must satisfy the following nonlinear equations,

$$u^k = \mathbf{v}^T \boldsymbol{\sigma}[\mathbf{W}\mathbf{y}^k + \mathbf{d}], \quad k = 1, \ldots, p \tag{5}$$

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

$$\mathbf{u} = [u^1 \ \cdots \ u^k]^T \tag{6}$$

eq. 5 can be written using matrix notation:

$$\mathbf{u} = \mathbf{S}\mathbf{v} \tag{7}$$

S is a matrix of sigmoidal functions evaluated at input-to-node values, n_i^k, each representing the magnitude of the input-to-node variable, n_i, to the i^{th}-node for the training pair, k:

$$\mathbf{S} \equiv \begin{bmatrix} \sigma(n_1^1) & \sigma(n_2^1) & \cdots & \sigma(n_s^1) \\ \sigma(n_1^2) & \sigma(n_2^2) & \cdots & \sigma(n_s^2) \\ \vdots & \vdots & \ddots & \vdots \\ \sigma(n_1^p) & \sigma(n_2^p) & \cdots & \sigma(n_s^p) \end{bmatrix} \tag{8}$$

Therefore, the nonlinearity of these equations arises purely from the implicit dependence of the nonlinear function's argument on the input neural parameters.

The novel technique is based on the idea that if the input-to-node values, n_i^k, are known, **S** is a known matrix, and eq. 7 becomes a linear system of equations to be solved for **v**. When the number of nodes, s, is chosen equal to the number of training pairs, p, **S** is square. If it also is non-singular, eq. 7 is a full-rank linear system for which an exact solution always exists. The input parameters affect the solution of the output initialization equations only through the input-to-node values and determine the nature of **S**. One of the strategies [4] that can produce a well-conditioned **S** consists of generating the input weights according to the following rule,

$$w_{ij} = f r_{ij} \tag{9}$$

where r_{ij} is chosen from a normal distribution with zero mean and unit variance that is obtained using a random number generator. The scalar f is arbitrary and of order $O(10)$; it can be slightly varied based on how closely spaced the training pairs are. The input bias, **d**, is computed to center each sigmoid at one of the training pairs, $\{\mathbf{y}^k, u^k\}$,

$$\mathbf{d} = -\mathbf{Y}\mathbf{W}^T \tag{10}$$

where **Y** is a matrix composed of all the input elements from the training set:

$$\mathbf{Y} \equiv [\mathbf{y}^1 \ \cdots \ \mathbf{y}^k]^T \tag{11}$$

This is equivalent to distributing the sigmoids across the input space as suggested by the Nguyen-Widrow initialization algorithm [5], except here it is achieved simply by solving a linear system (eq. 10) for the network's input bias.

The input elements, $\mathbf{y}^k$, from the training set are normalized, and **d** is computed based on the input weights, according to eq. 10. Thus, the scaling factor, f, scales the distribution of the input-to-node values, establishing their order of magnitude. While p sigmoids are centered, the remaining sigmoids come close to being saturated for inputs whose absolute value is greater than 5. A variance of order $O(10)$ allows a good fraction of the sigmoids to be highly saturated, contributing to a smooth approximating function and producing a non-singular **S**.

2.1 Incorporating Partial Derivatives in Simply Connected Neural Network Training

The approach previously introduced has proven that there exist many p-node networks capable of fitting the input/output training set exactly. Using derivative information during training can improve the network's generalization properties. Suppose the training set, $\{\mathbf{y}^k, u^k, (\partial h/\partial \mathbf{y})|_{\mathbf{y}^k}\}_{k=1,\ldots,p}$, consists of all partial derivatives of the function $h(\bullet)$ with respect to its inputs along with the input/output samples. That is, the following vectors are known:

$$\mathbf{c}^k \equiv \left(\left. \frac{\partial h}{\partial \mathbf{y}} \right|_{\mathbf{y}^k} \right)^T, \quad k = 1, \ldots, p \tag{12}$$

The gradient, $\partial h / \partial \mathbf{y}$, is defined as a row vector.

The partial derivatives of the neural network's output with respect to its inputs correspond to the elements of the known gradient. Therefore, in addition to the output equations (eq. 7), the parameters also must satisfy the *gradient initialization equations,*

$$\left(\mathbf{c}^k\right)^T = \mathbf{W}^T \left\{ \mathbf{v} \otimes \boldsymbol{\sigma}'\left[\mathbf{n}^k\right] \right\} \tag{13}$$

where the symbol "$\otimes$" denotes element-wise vector multiplication, and:

$$\mathbf{n}^k = \mathbf{W}\mathbf{y}^k + \mathbf{d}, \quad k = 1, \ldots, p \tag{14}$$

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

$$\boldsymbol{\sigma}'[\mathbf{n}] \equiv [\sigma'(n_1) \ \ldots \ \sigma'(n_s)]^T \tag{15}$$

The input weights, $\mathbf{W}$, appear explicitly in the gradient equations. If the number of nodes, s, is chosen equal to the number of training triads, p, the number of input weights equals the number of gradient equations. When input-to-node values are known, eq. 7 can be solved for the output weights; subsequently, eq. 13 can be written as the following linear systems to be solved for $\mathbf{W}$,

$$\left(\mathbf{c}^k\right)^T = \mathbf{B}^k \mathbf{W}, \tag{16}$$

where:

$$\mathbf{B}^k \equiv [v_1\sigma'(n_1^k) \ \ v_2\sigma'(n_2^k) \ \ \ldots \ \ v_s\sigma'(n_s^k)] \tag{17}$$

It is possible to solve both output and gradient equations exactly when the dimension of the inputs for which the gradient is unspecified is equal to p or in other special cases particularly relevant to control applications, such as neural modeling of gain-scheduled controllers [4].

In the general case, it is found that a suitable way to incorporate the gradient equations in the training process is to use eq. 16 to obtain a more stringent criterion of formation for the input weights. A first estimate of the output weights, $\mathbf{v}$, and of the input-to-node values, n_i^k, to be used in eq. 16 can be obtained from the solution of the output equations, eq. 7, based on the randomized $\mathbf{W}$. 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 steps node-by-node update algorithm. The underlying concept is that the input bias, d_i, and the input-to-node values associated with the i^{th} node,

$$\mathbf{n}_i \equiv \left[n_i^1 \ \cdots \ n_i^k\right]^T \tag{18}$$

can be computed solely from the input weights associated with it:

$$\mathbf{w}_i \equiv \left[w_{i1} \ \cdots \ w_{iq}\right]^T \tag{19}$$

At each step, the i^{th} sigmoid is centered at the k^{th} training pair through the input bias d_i, i.e., $n_i^k = 0$, when $i = k$. The k^{th} gradient equations are solved for the input weights associated with the i^{th} node, i.e., from eq. 16:

$$v_i\sigma'\left(n_i^k\right)\mathbf{w}_i = \mathbf{c}^k - \begin{bmatrix} \sum_l v_l\sigma'\left(n_l^k\right)w_{l1} \\ \vdots \\ \sum_l v_l\sigma'\left(n_l^k\right)w_{lq} \end{bmatrix}, \tag{20}$$

$l = 1,\ldots, (i-1), (i+1),\ldots, p$ and $l \neq i$

The remaining variables are obtained from the initial estimate of the weights. The i^{th} input bias is computed individually,

$$d_i = -\mathbf{y}^k \mathbf{w}_i \tag{21}$$

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

$$\mathbf{n}_i = d_i + \mathbf{Y}\mathbf{w}_i \tag{22}$$

At the end of each step, eq. 7 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 assumption is that the i^{th} node input weights mainly contribute to the k^{th} partial derivatives, $\mathbf{w}_i$, because the i^{th} sigmoid is centered 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_{ij} 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 initialization equations can be solved exactly for the network's parameters.

3. Extension to Fully Connected Feedforward Neural Networks

Fully connected feedforward neural networks, also referred to as ordered neural networks, are a class of computational networks that allow for inner-layer connections, but avoid recurrence. The connections among neurons are ordered, such that the information flows continuously from the network's inputs to its output. Neurons are numbered sequentially and any neuron is connected only to higher-numbered neurons. Outer (i.e., linear) connections between the networks' inputs and output also are allowed; they are not treated here because including them would be a trivial extension of what follows. A typical ordered architecture having s nodes and q inputs is shown (Fig. 2). The inner-layer connections are drawn with a dashed line.

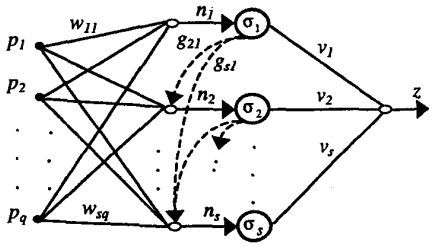

Figure 2. Example of a scalar-output fully connected feedforward network (input biases and summation symbols are omitted for simplicity).

The simply connected neural network (Fig. 1) is a special case of this ordered architecture. The newly introduced parameters, g_{ij}, represent the magnitude of the connection from the j^{th} to the i^{th} neuron, thus $j < i$.

The input to each neuron, n_i, now includes the input coming from lower numbered neurons. Thus, the vector $\mathbf{n}$ of input-to-node variables is written in terms of the input parameters, $\mathbf{W}$ and $\mathbf{d}$, as well as the inner-layer weights, $\mathbf{G}$,

$$\mathbf{n} = \mathbf{G}\boldsymbol{\sigma}[\mathbf{n}] + \mathbf{W}\mathbf{p} + \mathbf{d} \tag{23}$$

where:

$$\mathbf{G} \equiv \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ g_{21} & 0 & 0 & \cdots & 0 \\ g_{31} & g_{32} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_{s1} & g_{s2} & \cdots & g_{s(s-1)} & 0 \end{bmatrix} \tag{24}$$

The output of the network is still of the form,

$$z = \mathbf{v}^T \boldsymbol{\sigma}[\mathbf{n}] \tag{25}$$

where the quantities previously introduced are defined consistently. The network partial derivatives acquire a more complicated form due to the inner-layer connections. The derivative of the i^{th} input-to-node variable with respect to a network input, p_j, is,

$$\frac{\partial n_i}{\partial p_j} = w_{ij} + g_{i(i-1)}\sigma'(n_{i-1})\frac{\partial n_{i-1}}{\partial p_j} + g_{i(i-2)}\sigma'(n_{i-2})\frac{\partial n_{i-2}}{\partial p_j} + \ldots + g_{i1}\sigma'(n_1)\frac{\partial n_1}{\partial p_j} \tag{26}$$

therefore:

$$\frac{\partial z}{\partial p_j} = \sum_{i=1}^{s} v_i\sigma'(n_i)\frac{\partial n_i}{\partial p_j} = \sum_{i=1}^{s} v_i\sigma'(n_i)w_{ij} + \sum_{i=1}^{s} v_i\sigma'(n_i)\left(\sum_{\ell=1}^{i-1} g_{i\ell}\sigma'(n_\ell)\frac{\partial n_\ell}{\partial p_j}\right) \tag{27}$$

If the gradient with respect to all of the inputs is known, $j = 1, \ldots, q$; if it is known with respect to e of the inputs, $j = 1, \ldots, e$, where $e < q$. In the latter case, the input weights corresponding to the remaining $(q-e)$ inputs do not appear explicitly in the gradient equations and play only an explicit role within the input-to-node equations (eq. 23).

The output initialization equations can be written as eq. 7. Under the assumption that the input-to-node values, n_i^k, are known, these equations also become linear. Hence, when $\mathbf{S}$ is non-singular, they always have an exact solution provided $s = p$. The gradient initialization equations are derived by equating the network partial derivatives (eq. 27) to the known gradients:

$$\left(\mathbf{c}^k\right)^T = \mathbf{B}^k\left\{\mathbf{W} + \mathbf{G}\left(\boldsymbol{\sigma}'[\mathbf{n}^k] \otimes \left.\frac{\partial \mathbf{n}}{\partial \mathbf{p}}\right|_k\right)\right\}, \quad k = 1, \ldots, p \tag{28}$$

The Jacobian $\partial\mathbf{n}/\partial\mathbf{p}$ is defined as an $s \times s$ matrix where the $(i,j)^{th}$ element consists of $\partial n_i / \partial p_j$. When $s = p$, the number of input weights, w_{ij}, is equal to the number of gradient equations. Here, these equations become linear when the inner-layer weights and the input-to-node values are both known. They cannot always be solved exactly, but a p-step algorithm can be devised to select input and inner-layer weights from gradient and input-to-node equations, respectively.

It is found that if none of the gradients are known, the inner-layer weights play no significant role and the network can be collapsed to the simply connected architecture. Hence, the approach helps identifying superfluous connec-

tions, based on the given training set, without resorting to iterative optimization techniques [6].

4. Algebraic Training of a Forward Neural Network

The proposed approach is implemented to train a simply connected neural network that approximates the trim map of a transport aircraft. The trim map represents an inversion of the aircraft model and, in series with the aircraft, it provides a feed-forward path [7]. This type of *forward neural network* finds application in the design of neural controllers that are motivated by linear control systems. Under perfect conditions of exact coincidence between the apriori model and the actual plant, the control is provided solely by the forward network. Perturbation feedback signals that compensate for inaccuracy in the aircraft model and for external disturbances also can be processed by neural networks [4].

It is assumed that a nonlinear differential equation describes the aircraft dynamics,

$$\dot{\mathbf{x}}(t) = \mathbf{f}[\mathbf{x}(t), \mathbf{p}(t), \mathbf{u}(t)] \tag{29}$$

For simplicity, a reduced-order longitudinal-axis model is considered, and disturbance effects and uncertainties in the parameters, $\mathbf{p}$, are ignored. The state vector, $\mathbf{x}$, consists of airspeed V, flight path angle γ, altitude $-z$, and pitch angle θ. The controls are throttle position δT and elevator δE; i.e., $\mathbf{u} = [\delta T \ \delta E]^T$.

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

$$\mathbf{u}_c = \mathbf{h}(\mathbf{x}_c, \mathbf{p}) = \mathbf{h}(\mathbf{y}_c) \tag{30}$$

such that,

$$\mathbf{f}(\mathbf{x}_c, \mathbf{p}, \mathbf{u}_c) = \mathbf{f}[\mathbf{x}_c, \mathbf{p}, \mathbf{h}(\mathbf{y}_c)] = \mathbf{0} \tag{31}$$

with $\mathbf{x}_c$ computed from $\mathbf{y}_c$ and from the flight conditions $\mathbf{p}$, as in [4]. The aircraft trim map, U_c, is obtained by solving the steady-state equation (eq. 31) numerically p times over the aircraft's operational domain $\Im$:

$$U_c(\mathbf{x}_c, \mathbf{p}) \equiv \left\{ \mathbf{u}_c^k : \left(\mathbf{x}_c^k, \mathbf{p}^k\right) \in \Im, \ \mathbf{f}\left(\mathbf{x}_c^k, \mathbf{p}^k, \mathbf{u}_c^k\right) = \mathbf{0}, \ k = 1, \ldots, p \right\} \tag{32}$$

Local gradients of this hypersurface can be obtained from the linearized equations of motion [4] at each set point $(\mathbf{x}_c^k, \mathbf{u}_c^k)$, such that the following is known:

$$\mathbf{C}_F^k \equiv \left(\left. \frac{\partial \mathbf{h}}{\partial \mathbf{y}_c} \right|_{\mathbf{y}_c^k} \right)^T, \ k = 1, \ldots, p \tag{33}$$

The forward neural network, $\mathbf{NN}_F$, is trained using a set of trim data, $\left\{ \mathbf{y}_c^k, \mathbf{u}_c^k, \mathbf{C}_F^k \right\}_{k=1,\ldots,p}$, that reflects the nonlinear characteristics of the aircraft.

The vector output, $\mathbf{u}_c$, is produced by two scalar, simply connected networks with input $\mathbf{y}_c = [V \ \gamma]^T$:

$$\begin{bmatrix} \delta T_c \\ \delta S_c \end{bmatrix} = \begin{bmatrix} NN_{F_1}(\mathbf{y}_c) \\ NN_{F_2}(\mathbf{y}_c) \end{bmatrix} \tag{34}$$

Every row of $\mathbf{C}_F^k$ provides the gradient, $\mathbf{c}^k$, for a control element. In this example, the commanded airspeed and path angle vary between 300-460 Kt and 1°-2.8°, respectively, and the altitude remains constant at 30,000 ft. Figure 3 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).

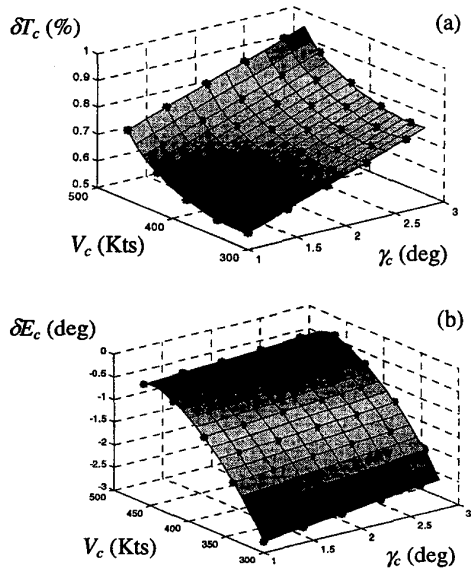

Figure 3. Trim map control surfaces and corresponding training samples.

45 samples are used to train the neural networks. They are symbolized by asterisks superimposed on the corresponding surfaces (Fig. 3). Therefore, each network contains 45 nodes.

The weights of NN_{F_1} and NN_{F_2} are determined from the initialization equations. The gradient tolerances are set at 0.05 %/Kt $(\partial(\delta T_c)/\partial V_c)$, 0.5 %/deg $(\partial(\delta T_c)/\partial \gamma_c)$, 0.0008 deg/Kt $(\partial(\delta E_c)/\partial V_c)$, and 0.05 deg/deg $(\partial(\delta E_c)/\partial \gamma_c)$. For NN_{F_1}, the parameters obtained from

the output equations produce a lumpy surface (Fig. 4), and the gradient tolerances are not immediately satisfied.

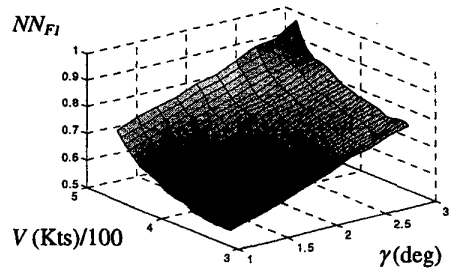

Figure 4. Trim throttle function approximation obtained from output initialization equations alone.

The weights are further refined using the p-step gradient algorithm, finally producing the output surface in Fig. 5a. For NN_{F_2}, the weights obtained from the output initialization equations alone already satisfy the gradient tolerances; therefore, the final parameters (Fig. 5b) are obtained in only one step.

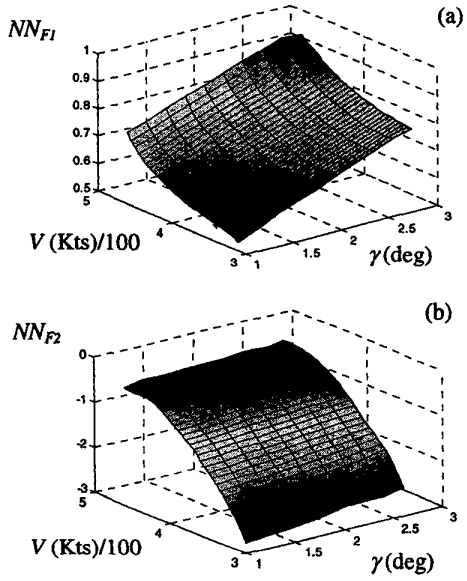

Figure 5. Final trim control function approximation; (a) is obtained from output and gradient initialization equations, (b) is obtained from output initialization equations.

The neural output surfaces are plotted over a fine-grid input space, to demonstrate the networks' interpolation abilities. The approximating function (Fig. 5) could be improved by running the p-step algorithm again with smaller gradient tolerances, or by increasing the number of training pairs, p. The training time is remarkable (a MATLAB code trained a 45-node network in 0.345 sec) and, in some cases, it is even less than the networks' input/output execution time.

5. Conclusions

A novel algebraic technique that trains computational neural networks and affords a great deal of insight into both neural approximation properties and neural control applications is developed. The relevant principles and the overall approach are described for two feedforward architectures. The method allows input/output training data to be matched exactly and gradient information to be incorporated in the training process simply by solving linear algebraic systems of equations for adjustable network parameters. A forward neural network is trained algebraically to approximate a transport aircraft's trim map. The results show that training is remarkably fast and straightforward, and that the generalization and interpolation capabilities of the networks are preserved.

Acknowledgement

This research has been supported by the Federal Aviation Administration and the National Aeronautics and Space Administration under FAA Grant No. 95-G-0011.

References

[1] Barron, A. R. (1993) "Universal Approximation Bounds for Superposition of a Sigmoidal Function," *IEEE Transactions on Information Theory*, V 39, no. 3, pp. 930-945.

[2] Kolmogorov, A. N. (1957) "On the Representation of Continuous Functions of Several Variables by Superposition of Continuous Functions of One Variable and Addition," *Dokl. Akad. Nauk SSSR*, V 114, pp. 953-956.

[3] Hunt, K. J., Sbarbaro, D., Zbikowski, R., Gawthrop, P. J. (1992) "Neural Networks for Control Systems – A Survey," *Automatica*, V 28, no. 6, pp. 1083-1112.

[4] Ferrari, S., Stengel, R. F., (2000) "Classical/Neural Synthesis of Nonlinear Control Systems," *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, August 14-17, Denver, CO.

[5] Nguyen, D., Widrow, B. (1990) "Improving the Learning Speed of 2-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights," *Proc. Intl. Joint Conf. on Neural Networks*, San Diego, CA, V III, pp. 21-26.

[6] KrishnaKumar, K. (1993) "Optimization of the Neural Net Connectivity Pattern Using a Backpropagation Algorithm," *Neurocomputing*, V 5, pp.273-286.

[7] Cicolani, L. S., Sridhar, B., Meyer, G. (1979) "Configuration Management and Automatic Control of an Augmentor Wing Aircraft with Vectored Thrust," NASA Technical Paper, TP-1222.