

# Verification and Validation of Neural Networks for Safety-Critical Applications

Jason Hull & David Ward  
Barron Associates, Inc.  
1160 Pepsi Place, Suite 300  
Charlottesville, VA 22901

[hull@barron-associates.com](mailto:hull@barron-associates.com), [ward@barron-associates.com](mailto:ward@barron-associates.com)

Radosław R. Zakrzewski  
Goodrich Aerospace, Fuel and Utility Systems  
100 Pantan Rd., Vergennes, VT 05491, USA  
[Radek.Zakrzewski@goodrich.com](mailto:Radek.Zakrzewski@goodrich.com)

## Abstract

Onboard nonlinear models are a key enabling technology for virtual sensors, model-based control, reconfigurable control and model-based diagnostic algorithms. Before such models can be used in safety-critical applications, such as civilian aircraft, they must undergo extensive testing to verify that there is no combination of inputs that will generate an undesirable output. While this can be done relatively easily for lookup tables and some nonlinear decision logic, it is more difficult for more powerful models such as multilayer perceptrons (MLPs) or polynomial neural networks (PNNs). This paper presents analysis techniques that can be used as part of a verification procedure for PNNs that are trained to replace lookup tables in a variety of safety-critical control applications. The technique builds on previous research that uses Lipschitz constants to provide guaranteed bounds on network output and error for all possible inputs without having to test the network at all possible input combinations. The focus of the work presented here is on static, feed-forward, multilayer networks, with polynomial basis functions. The methods described form the basis of a software tool, which is in the process of being qualified by the FAA for use in verifying neural networks for safety-critical flight control applications.

## 1 Introduction

Many current-generation flight-critical systems employ lookup tables to map a set of independent variables (e.g., angle-of-attack and dynamic pressure) to a dependent variable (e.g., a control gain or system parameter); however, lookup tables are only effective when the mapping is relatively simple and there are limited independent variables. There is currently an explosion of interest in replacing lookup tables with fixed-structure neural network models for near-term upgrades of current-generation aviation systems. In addition to offering improved modeling accuracy, computational efficiency, and reduced development costs, neural network models have a number of safety advantages. For current-generation systems, these models can result in a significant reduction in software complexity, and the number of hardware components required to perform state-of-the-art diagnostics and control, leading to more reliable systems.

In spite of the advantages offered by onboard models, and in spite of their widespread and successful use in other controls applications, their use in aircraft, especially commercial aircraft, has been limited. This is due, in part, to the long development cycle of an average jetliner; however, safety concerns are primarily to blame. Any nonstandard technique is thoroughly examined by certification authorities before it is approved for use in aircraft - a process that takes time. Therefore, absence of neural nets in aircraft software only 15 years after their revival is perhaps not as surprising as it may seem.

Moreover, and more to the point for this paper, there is currently no systematic way to verify and validate (V&V) neural networks for use in an aircraft flight-critical system. V&V guidelines require a guarantee of the maximum network *error* and *output* over the range of all possible inputs. This paper documents recently developed and broadly applicable analysis methods that provide such guarantees, thus allowing the safe usage of nonlinear onboard models in certifiable systems.

This paper begins by summarizing a method for verifying and validating single-layer perceptrons [1]. This method is then extended with application to other forms of nonlinear function approximation, particularly single- and multiple-layer networks employing polynomial basis functions such as PNNs, Pi-Sigma networks, and polynomial models resulting from orthogonal basis function modeling. In this method, as in [1], we consider the case of a neural network replacing a known functional mapping, specifically a lookup table employing multilinear interpolation. We provide an alternate method for bounding the output of such a lookup table and leverage this technique to bound the approximation error between the lookup table and the neural network. It should also be noted that the validation method is independent of the method used to generate the onboard model (i.e., connections, weights, and basis functions); however, it assumes the structure of the model will *not* change in flight (as is done in some adaptive neural network flight control approaches).

## 2 Background and Underlying Assumptions

Before discussing the steps required to validate a neural network for a safety-critical application, it is important to understand the nature of the networks that are being addressed, and how they are or are not like technologies currently employed in *certified* safety-critical systems.

1. *The networks are static maps.* The structure or parameters in the networks used here are *not* allowed to adapt in time. In this way, the networks are like a lookup table that maps a set of measured independent variables into the required dependent variable(s).
2. *The networks are nonlinear maps.* Like a lookup table, the networks are designed to interpolate between known points (which make up the synthesis/training database); however, unlike most lookup tables, this interpolation is nonlinear. Therefore, one needs a method to ensure that the network output is reasonably bounded.
3. *The networks are developed in much the same way as nonlinear elements currently employed in certified systems.* There is a strong similarity between the semiautomatic process of developing neural networks and the manual process currently employed for developing certain nonlinear

control law elements that *can* be validated. In the manual process, engineers use their knowledge of the behavior of the system under known conditions to construct a map between observables and a desired intermediate variable. If, during testing, the map does not work well at a given input condition, the logic is modified. In the same way, a neural network is synthesized on a given set of data. If during testing the network fails to perform well at a given input condition (most likely one very different from those used to synthesize the network), the network is retrained with the new data now included as part of the synthesis database.

Given that the networks are similar to commonly used components, one would expect the certification process to be similar as well. In fact, it turns out that the only difficult question regarding certification of onboard neural networks is, "are there some untested inputs or unforeseen input combinations that will result in an unexpected (and problematic) output?" In the case of continuous input variables, there is an infinite set of input values; therefore, it is not feasible to check every possible set of input values prior to operation. This, in and of itself, is not problematic. Consider a lookup table (e.g., gain schedule); here, it is not possible to test all input/output combinations. However, because one uses a known method of interpolation in a lookup table, it is guaranteed that an interpolated value is bounded by the values of its neighbors. For nonlinear functions, including neural networks, bounds on interior values are difficult to guarantee. Even if network output is limited for a large set of test points, it is not guaranteed to be limited for new inputs in the range of the tested region. Yet such a performance guarantee is essential if one is to use neural networks in safety critical applications.

### 3 Verification of a Single-Layer Perceptron

Deterministic verification techniques for complex nonlinear models have already been developed to provide guaranteed output and error bounds for a single-layer perceptron employing a sigmoidal basis function [1]. This proven method combines exhaustive testing with an analysis technique that guarantees the network behavior within the regions between test points.

To find this bound, first the verification region is divided into a volume of *rectangular* test cells. The model is then evaluated at each cell vertex and the network output and approximation error is found at each point. A bound on the network output within the untested rectangular regions is then found using the maximum growth rate for the network in a first order Taylor series expansion. A bound on the network approximation error is found using the maximum second order growth rate of the network and lookup table it is replacing in a second order Taylor series expansion. These maximum first and second order bounds on the growth rate are Lipschitz constants,  $K_f$  and  $K'_f$ , that satisfy the inequalities

$$\|f(\underline{v}) - f(\underline{z})\| \leq K_f \|\underline{v} - \underline{z}\| \quad (1)$$

and,

$$\|f'(\underline{v})^T - f'(\underline{z})^T\| \leq K'_f \|\underline{v} - \underline{z}\| \quad (2)$$

over the entire test cell.

The constants that satisfy these inequalities when  $f$  is a multi-layer perceptron defined by

$$f(x) = w_0^{(o)} + \sum_{i=1}^p w_i^{(o)} \gamma \left( w_{i,0}^{(h)} + \sum_{j=1}^n w_{i,j}^{(h)} x_j \right) \quad (3)$$

where  $\gamma$  is the hyperbolic tangent function, are derived in [1]. The Taylor series expansions that provide network output, and error bounds are also derived in [1] and are repeated in equations (4) through (7) for convenience.

$$M_{lower} = \bar{f} - K_f D_{max} \quad (4)$$

$$M_{upper} = \bar{f} + K_f D_{max} \quad (5)$$

$$E_{lower} = \bar{e} + G_{min} D_{max} - \frac{1}{2} (K'_f + K'_\phi) D_{max}^{(2)} \quad (6)$$

$$E_{upper} = \bar{e} + G_{max} D_{max} + \frac{1}{2} (K'_f + K'_\phi) D_{max}^{(2)} \quad (7)$$

In these expansions,  $\bar{f}$  and  $\bar{e}$  are the mean values of the network output and approximation error over the vertices of the test cell,  $D_{max}$  and  $D_{max}^{(2)}$  are the maximum values of the mean distance, and distance squared, between an arbitrary point within the test cell and all the vertices,  $G_{max}$  and  $G_{min}$  are the maximum and minimum directional derivatives over the vertices and  $K_\phi$  and  $K'_\phi$  are the Lipschitz constants for the lookup table.

It is necessary to use Lipschitz constants to bound this type of network because the hyperbolic tangent function shown in equation (8) has an infinite number of non-zero derivatives.

$$\gamma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (8)$$

Determining an absolute bound for such a function without Lipschitz constants would require the calculation of the maximum and minimum directional derivatives out to infinite order – not a practical option. Lipschitz constants allow us to terminate this Taylor series at any order with a final term that bounds the growth contributed by all the higher order terms. The expansion-order is determined by making a tradeoff between computational complexity and the conservativeness of the bounds that one is willing to tolerate

### 4 Verification & Validation of Polynomial Neural Networks

The research presented in this paper is an extension of this verification technique to include static, multiple-layer, feed forward network structures employing polynomial basis functions. These include PNNs,  $\Pi\Sigma$  networks, and polynomial models resulting from orthogonal basis function modeling.

PNNs are compositions of Kolmogorov-Gabor (KG) multinomials [2], or algebraic sums of terms. The attention to algebraic elements stems from the pioneering work in the 1940s of Kolmogorov and, working independently, Gabor [3]. Each

demonstrated the near-universality of multinomials in representing physical processes, including dynamical systems. The KG multinomial,

$$y = a_0 + \sum_i a_i x_i + \sum_i \sum_j a_{ij} x_i x_j + \dots \quad (9)$$

can model any analytic single-valued transformation [4,5]. In fact, recent developments in statistics, information theory, computational methods, and approximation theory suggest that a multinomial description of the network learning process highlights the similarities among (as well as important differences between) adaptive network syntheses, backward-error propagation techniques, and modern statistical inference methods [6].

The types of networks that can be verified with the analysis techniques presented in this paper include static, feed-forward, multiple-layer structures with no internal node limiting. A typical PNN is shown in Figure 1 with each node implementing a unique KG multinomial of any degree. The only restriction imposed on the internal connections is that they are feed forward; each node can have any number of inputs and feed into any node between itself and the output node.

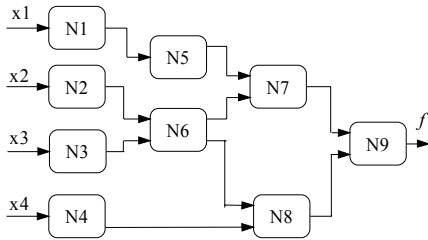


Figure 1 – Typical PNN Structure

$\Pi\Sigma$  networks and polynomial models resulting from orthogonal function modeling are each a subset of this general structure. The nodes for a  $\Pi\Sigma$  network are restricted to either add or multiply inputs, while the polynomial models collapse the PNN structure to a single multinomial. One property common to all these network types, however, is that they each have a finite number of non-zero derivatives. This property is leveraged in the V&V process to bound the output and error of these networks less conservatively without using Lipschitz constants.

The high-level verification approach for these types of networks is the same two-step process described above; (1) find the network output and approximation error over a uniform grid of test points, and (2) determine the output and error bounds within the untested cells. The difference lies in the method used to calculate these bounds.

#### 4.1 Network Output Bounds

Given that the network will have a finite number of nonzero derivatives, a finite order Taylor series expansion can be found at each vertex that gives the network output as a function of the distance from the vertex. If each term in the expansion is then bounded by the maximum directional derivative for its respective order, then this expansion provides an absolute bound on the network as a function of the distance from a test point.

To illustrate this, consider a PNN with  $m$  inputs and the maximum nonzero derivative order denoted as  $p$ . Beginning with a single vertex of an  $m$ -dimensional rectangular test cell, we seek a maximum and minimum bound on the output of the function at a test point,  $\tilde{x}$ , as it moves from the vertex into the interior of the cell along a straight line. Applying a Taylor series expansion about the vertex gives

$$f(\tilde{x}) = f(x_{vertex}) + \sum_{n=1}^p \frac{f^{(n)}}{n!} D(\tilde{x}, x_{vertex})^n \quad (10)$$

In this expansion,  $f^{(n)}$  is the  $n^{th}$  order derivative in the direction given by the unit vector

$$u = \frac{(\tilde{x} - x_{vertex})}{D(\tilde{x}, x_{vertex})}$$

where  $D(\tilde{x}, x_{vertex})$  is the distance from  $\tilde{x}$  to the vertex.

The first rate of change is given by the dot product of the gradient vector,  $G$ , and the unit vector  $u$ .

$$f' = G \cdot u \quad (11)$$

The second order and higher directional derivatives are given by

$$f^{(n)} = u^T H_f^{(n)} u \quad (12)$$

where  $H_f^{(n)}$  is the  $n$ -dimensional matrix of partial derivatives of  $f$  (e.g., an  $m \times m$  Hessian for the second derivative, or an  $m \times m \times m$  matrix for the third derivative).

The output of the function within the rectangle can then be bounded if the maximum and minimum values for the directional derivatives are used in each term of the complete Taylor series expansions

$$f(\tilde{x}) \leq f(x_{vertex}) + \sum_{n=1}^p \frac{f_{max}^{(n)}}{n!} D(\tilde{x}, x_{vertex})^n \quad (13)$$

$$f(\tilde{x}) \geq f(x_{vertex}) + \sum_{n=1}^p \frac{f_{min}^{(n)}}{n!} D(\tilde{x}, x_{vertex})^n \quad (14)$$

The maximum and minimum first order directional derivatives are given by the Euclidean norm of the gradient vector

$$-\|G\|_2 \leq f' \leq \|G\|_2 \quad (15)$$

The maximum and minimum values of the higher order directional derivatives are found by solving the optimization problems:

$$\min \{ u^T H_f^{(n)} u : \|u\| = 1 \} \leq f^{(n)} \leq \max \{ u^T H_f^{(n)} u : \|u\| = 1 \}$$

In this application  $H_f^{(n)}$  will always be a symmetric matrix, therefore the solution is given by the maximum and minimum

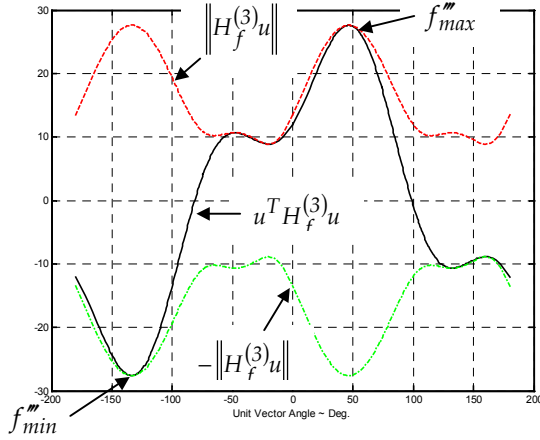
eigenvalues of  $H_f^{(n)}$  [7]. This is difficult to envision when  $H$  is of dimension greater than two, however, the solution is given by the maximum value of  $\|H_f^{(n)}u\|$  that satisfies

$$u^T H_f^{(n)} u = \|H_f^{(n)} u\| \quad (16)$$

and the solution to the minimum optimization problem is given by the minimum value that satisfies

$$u^T H_f^{(n)} u = -\|H_f^{(n)} u\| \quad (17)$$

which are the maximum and minimum eigenvalues. This is illustrated in Figure 2 for the third derivative of a two-input network.



**Figure 2 Third Order Directional Derivatives for a Two Input Polynomial Neural Network**

Solving these optimization problems, or calculating these eigenvalues of  $H_f^{(n)}$ , requires an iterative numerical procedure, which may hinder qualification of the V&V tool under DO-178B and other FAA directives. It can be shown, however, that for a symmetric matrix, the maximum and minimum eigenvalues are given by a convex function of the matrix elements. If this property is insufficient in terms of qualifying the tool, this calculation can be avoided by recognizing that for a symmetric matrix the singular values are the absolute values of the eigenvalues. This implies the following inequalities:

$$\sigma_{\max}(H_f^{(n)}) \geq \lambda_{\max}(H_f^{(n)}) \quad (18)$$

and,

$$-\sigma_{\max}(H_f^{(n)}) \leq \lambda_{\min}(H_f^{(n)}) \quad (19)$$

In these inequalities the maximum singular value,  $\sigma_{\max}(H_f^{(n)})$ , is the definition of the spectral norm of  $H_f^{(n)}$ . The spectral norm is

bounded by the Frobenius norm [7], which is a simple non-iterative calculation that can be part of an FAA-qualified tool. Therefore, for derivative order two and greater, the maximum and minimum directional derivative is bounded by the Frobenius norm of  $H_f^{(n)}$ :

$$f_{\max}^{(p)} = \|H_f^{(n)}\|_2 = \sqrt{\sum_{i_1, i_2, \dots, i_n=1}^m h_{i_1, i_2, \dots, i_n}^2} \quad (20)$$

$$f_{\min}^{(p)} = -\|H_f^{(n)}\|_2 = -\sqrt{\sum_{i_1, i_2, \dots, i_n} h_{i_1, i_2, \dots, i_n}^2} \quad (21)$$

A less conservative bound for the maximum first directional derivative can be found if  $u$  in equation (11) is limited to only those vectors pointing *into* the test cell. This is equivalent to solving the following optimization problem:

$$\max\{G \cdot u : \|u\| = 1, u_i > 0\}$$

If the signs of the elements of  $G$  are changed to re-define the positive direction as into the cell, the solution is given by:

$$f'_{\max} = \min(0, \max(g_i)) + \|G^+\| \quad (22)$$

where  $G^+$  is a vector of only the positive elements of the gradient vector. A less conservative minimum bound for the first directional derivative is found similarly giving

$$f'_{\min} = \max\left(0, \min(g_i)\right) - \|G^-\| \quad (23)$$

where  $G^-$  is a vector of only the negative elements of the gradient vector.

The maximum output within the entire test cell can now be bounded by applying the Taylor series expansion to each vertex of the cell. Adding these  $2^m$  inequalities and eliminating the dependency on  $\tilde{x}$  we obtain

$$f_{\max} = \bar{f} + \sum_{n=1}^p \frac{f_{\max}^{(n)}}{n!} \overline{D^n}_{\max} \quad (24)$$

where  $\bar{f}$  is the mean value of the function output, and  $f_{\max}^{(n)}$  is now the maximum directional derivative over all the vertices. The dependency on  $\tilde{x}$  is eliminated by determining the location that maximizes  $\overline{D^n}$ . For a rectangular test cell, this is given by

$$\overline{d^n}_{\max} = \frac{1}{2^m} \sum_{i_1=0}^1 \sum_{i_2=0}^1 \dots \sum_{i_m=0}^1 \left( \sqrt{\sum_{j=1}^m i_j \Delta_j^2} \right)^n \quad (25)$$

where  $\Delta_j$  is the length of the  $j^{\text{th}}$  side of the hyper-rectangle. The minimum output bounds within the cell are found similarly as

$$f_{min} = \bar{f} + \sum_{n=1}^p \frac{f_{min}^{(n)} \overline{D^n}_{max}}{n!} \quad (26)$$

## 4.2 Network Error Bounds

To determine the error bounds we consider the case where a network replaces a multidimensional lookup table. Using the same approach presented in the previous section, we can bound the output of such a mapping and ultimately the difference between the network and lookup table outputs.

### 4.2.1 Multi-Dimensional Lookup Table Mapping

While it may not be practical or desirable to implement in an actual flight-critical system, one can create a hypothetical multidimensional lookup table defined over an  $m$ -dimensional rectangle  $R = \langle \underline{x}^{(lo)}, \underline{x}^{(up)} \rangle$ , with each coordinate discretized into  $N_i - 1$  intervals of length  $\Delta_i$ . If we denote the entries of the table by  $\Phi(k_1, k_2, \dots, k_m)$ , then the output of the lookup table for an input vector,  $\underline{x}$ , that satisfies

$$x_i^{(lo)} + k_i \Delta_i \leq x_i \leq x_i^{(lo)} + (k_i + 1) \Delta_i \quad \text{for } i=1, 2, \dots, m$$

is defined by

$$\begin{aligned} \varphi(x) &= \frac{1}{\prod_{i=1}^m \Delta_i} \sum_{j_1=0}^1 \dots \sum_{j_m=0}^1 \Phi(k_1 + j_1, \dots, k_m + j_m) \left( \prod_{i=1}^m \delta_i^{j_i} \right) \\ \delta_i^0 &= x_i - (x_i^{(lo)} + k_i \Delta_i) \\ \delta_i^1 &= (x_i^{(lo)} + (k_i + 1) \Delta_i) - x_i \end{aligned}$$

When expanded, this function takes on a multinomial form consisting of linear elements and cross terms, and, like the PNNs in the previous section, will have a finite number of nonzero derivatives. If we denote the matrix of partial derivatives for each order,  $n$ , as  $H_\varphi^{(n)}$ , the set of off-diagonal elements,  $\{H_\varphi^{(n)}(i_1, i_2, \dots, i_n) | i_1 \neq i_2 \neq \dots \neq i_n\}$ , is given by

$$\frac{\partial \varphi}{\partial x_{i_1} \partial x_{i_2} \dots \partial x_{i_n}} = \sum_{j_1=0}^1 \sum_{j_2=0}^1 \dots \sum_{j_m=0}^1 \prod_{q=1}^n (1 - 2j_{i_q}) \Phi(k_j) \prod_{r=1}^m \delta_r^{j_r} \quad \begin{matrix} r \neq (i_1, i_2, \dots, i_n) \end{matrix}$$

where  $\Phi(k_j) = \Phi(k_1 + j_1, k_2 + j_2, \dots, k_m + j_m)$

With only linear elements and cross terms in  $\varphi(x)$ , the diagonal elements of  $H_\varphi^{(n)}$  will always be zero and the maximum value of  $n$  will equal the number of inputs.

### 4.2.2 Error Bounds

Applying the techniques presented in section 4.1 to the  $n^{\text{th}}$  order matrices of partial derivatives of the error function,

$$H_e^{(n)} = H_f^{(n)} - H_\varphi^{(n)} \quad (27)$$

the *difference* between the network and outputs of the hypothetical lookup table within a rectangular test cell are bounded by

$$e_{max} = \bar{e} + \sum_{n=1}^p \frac{e_{max}^{(n)} \overline{D^n}_{max}}{n!} \quad (28)$$

$$e_{min} = \bar{e} + \sum_{n=1}^p \frac{e_{min}^{(n)} \overline{D^n}_{max}}{n!} \quad (29)$$

where  $\bar{e}$  is the mean error over the  $2^m$  vertices, and  $e_{max}^{(n)}$  and  $e_{min}^{(n)}$  are the maximum and minimum  $n^{\text{th}}$  order directional derivatives of the error function.

## 4.3 Practical Issues

The analysis techniques proposed above require the calculation of all the non-zero partial derivatives at each test point. It can be shown that the number of test points for each cell grows exponentially with the number of inputs,  $2^m$ , and the number of non-zero partial derivatives grows exponentially as a function of the number of inputs and the maximum network degree. More precisely, this number is given by

$$d = \sum_{n=1}^p m^n \quad (30)$$

However, because  $H_f^{(n)}$  is symmetric, only

$$d = \sum_{n=1}^p \frac{(m-1+n)!}{(m-1)! n!} \quad (31)$$

elements are unique. Consider a six-input, two-layer network with a maximum node degree of three. Such a network will have 12,093,234 non-zero partial derivatives, but only 5,004 are unique. Thus, if one knew the number of times each partial derivative is repeated, the V&V software tool could be set up to compute and store only the unique derivatives. It turns out that one can compute *a priori*, the number of times a partial derivative appears in an  $n$ -dimensional symmetric matrix by

$$\frac{n!}{k_1! k_2! \dots k_m!} \quad (32)$$

where  $k_i$  represents the number of times the network is differentiated with respect to the  $i^{\text{th}}$  input. For example, the partial derivative  $\frac{\partial f}{\partial x_1 \partial x_1 \partial x_2 \partial x_3}$  for a 4 input network is repeated

$$\frac{4!}{2!1!1!0!} = 12 \text{ times.}$$

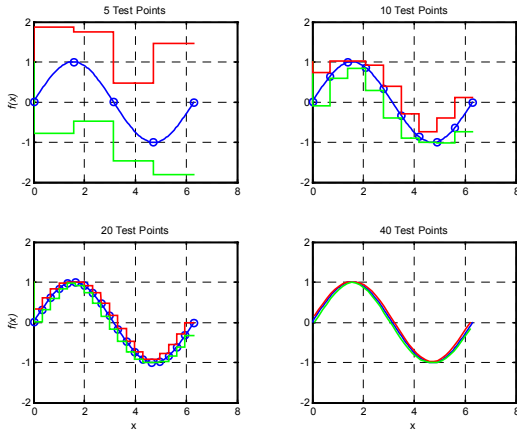
## 4.4 Experimental Results

The analysis techniques presented in this paper are currently being encoded into a software tool intended to automate the V&V process. Here we present a simple example using this tool to generate *output* bounds for a single input neural network. To illustrate these techniques for a highly nonlinear model we

created a network to estimate a cycle of the sine wave. The output of this function is given by

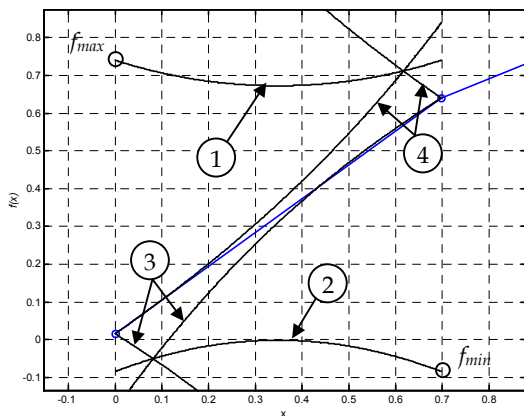
$$f(x) = -0.006x^5 + 0.089x^4 - 0.401x^3 + 0.285x^2 + 0.862x + 0.016$$

The maximum and minimum output bounds are shown below in Figure 3 for various grid dimensions. As equations (24) through (26) indicate the quality of the bounds is a function of this grid spacing. That is, the tighter grid spacing, the less conservative bounds.



**Figure 3 Network Output Bounds**

In Figure 4 we show exactly how these analysis techniques are applied to bound the output between two adjacent test points for this network. At each test point the Taylor series expansions given in equations (13) and (14) are shown (items 3 and 4). The intersection of these two functions provides a bound; however, this can be a difficult point to find, especially in a higher dimensional space. To avoid this search, we sum these expansions to get the mean output (items 1 and 2); this preserves the original inequalities of equations (13) and (14) for any point within the region. The maximum and minimum values of these functions, which are given by equations (24) and (26), occur at the end points given by  $D^n_{max}$ .



**Figure 4: Bounding Function Between Two Test Points**

This tool has also been applied to more complex multiple layer network structures being developed for civilian aircraft and

military rotorcraft applications. These networks had as many as six inputs and a maximum polynomial degree of nine, and it was possible to bound the output of a test cell to within 1-2% of the maximum output occurring at a vertex.

## 5 Conclusion

This paper presents an analysis technique that can provide guaranteed output and error bounds for neural networks that employ polynomial basis functions. This answers the question “is there some combination of inputs that results in an undesirable or unexpected output?”. If such a condition exists, this approach will show exactly where in the input space the trouble occurred. In such a case, a design engineer could fill in this region with more data and retrain the network. If a network is well behaved throughout the operating region, this tool will prove it certifiable for safety critical flight applications.

We considered only the case of a neural network replacing a known functional mapping, specifically a lookup table that could have been certified using conventional procedures. This technique could be extended to other functional mappings if we can bound their output. This may include the use of Lipschitz constants as in [1].

[1] R.R.Zakrzewski, "Verification of a trained neural network accuracy", Proc. International Joint Conf. on Neural Networks", Washington, DC, 2001, pp. 1657-1662.

[2] Barron, R., Mucciardi, A., Cook, F., Craig, J., and Barron, A., *Self-Organizing Methods in Modeling: GMDH Type Algorithms*, S.J. Farlow (ed.), Marcel Dekker, New York, NY, pp. 25-65, 1984.

[3] Gabor, D., Wilby, P., and Woodcock, R., “A Universal Nonlinear Filter, Predictor, and Simulator which Optimizes itself by a Learning Process,” *J.IEE*, received 1959.

[4] Ivakhnenko, A., "Polynomial Theory in Complex Systems," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-1, No. 4, pp. 364-378, 1971.

[5] Gabor, D., "Communication Theory and Cybernetics," *Transactions of IRE*, Vol. CT-1, No. 4, pp. 19, 1954.

[6] Barron, A. and Barron, R., "Statistical Learning Networks: A Unifying View," *Proceedings of the 20<sup>th</sup> Symposium on the Interface: Computing Science and Statistics* (Reston, VA), pp. 1313-1320, 1988.

[7] Horn, Roger A., Johnson, Charles R., “Matrix Analysis” Cambridge University Press, 1985.