JOURNAL OF SPACECRAFT AND ROCKETS
Vol. 31, No. 6, November–December 1994

Space Shuttle Main Engine Sensor Modeling Using Radial-Basis-Function Neural Networks

Kevin R. Wheeler* and Atam P. Dhawan†
University of Cincinnati, Cincinnati, Ohio 45221-0030
and
Claudia M. Meyer‡
Sverdrup Technology, Inc., Cleveland, Ohio 44212

An efficient method of parameter prediction is needed for sensor validation of Space Shuttle main-engine (SSME) parameters during real-time safety monitoring and post-test analysis. Feedforward neural networks (FFNN) have been used to model the highly nonlinear and dynamic SSME parameters during startup. Due to several problems associated with the use of feedforward networks, radial-basis-function neural networks (RBFNN) were investigated in modeling SSME parameters. In this paper, RBFNNs are used to predict the high-pressure oxidizer turbine discharge temperature, a redlined parameter, during the startup transient. Data from SSME ground test firings were used to train and validate the RBFNNs. The performance of the RBFNN model is compared with that of a FFNN model, trained with the Quickprop learning algorithm. In comparison with the FFNN model, the RBFNN-based model was found to be more robust against variations in architecture and network parameters, and was faster to train. In addition, the performance of the RBFNN model during nominal operation and during simulated input sensor failures was found to be robust in the presence of small deviations in the input.

Nomenclature

c_i = centroid (mean) of cluster i
F = matrix of activations
K = total number of clusters (basis functions) in system
w_i = value of weighted connection from unit i to output summation unit
σ_i^2 = variance of cluster i
$\phi()$ = activation function
°R = degrees Rankine

Introduction

REAL-TIME safety and post-test diagnostic systems for the Space Shuttle main engine (SSME) require automated sensor validation in order to prevent erroneous engine shutdowns or diagnoses. Current efforts to develop an automated sensor validation system for the SSME are based on the use of analytical redundancy techniques in which actual sensor data are compared with model predicted values.[1–5] Bayesian probability theory has been used to combine the information from a complete network of sensors and models into a single solution describing the health of each sensor under consideration.[1] In the event of a sensor failure, analytical redundancy techniques provide a predicted value for continued monitoring. Engine characteristic equations and first- and third-order empirical correlations can be used to successfully model a large number of parameters during steady-state operation of the SSME, but more complex models are required for parameters during the highly nonlinear startup transient.

It has been shown that feedforward neural networks with one hidden layer can uniformly approximate any continuous function.[6–8] A hidden layer of a network is any layer between the input and output layers. Feedforward neural networks (FFNN) trained with the backpropagation (BP) algorithm[9] have been used to model critical parameters during the SSME startup transient.[4] The neural-network models were used to predict a specific sensor value based upon a select set of inputs provided through other sensor measurements. The difference between the predicted and measured sensor values from a number of models can be used to indicate the health of the sensors and engine. There are however, several problems associated with feedforward neural networks trained with the BP algorithm.

First, BP neural networks are prone to getting stuck in local minima on the error surface, depending on the selection of network parameters such as learning rate, weight initialization, and architecture. This problem may be exacerbated by noise in the data. Another problem involves the difficulty in determining a minimal but adequate architecture that minimizes training time and optimizes generalization. The traditional approach has been trial and error. Finally, BP learning is a slow training algorithm if computed serially on a single-processor machine. The learning becomes even slower if the input dimension, the number of training vectors, or the number of hidden units becomes large. These issues are crucial for the SSME application, since the potential input dimension and number of training vectors is very large.

Radial-basis-function neural networks (RBFNN) are proposed as an alternative to BP-trained FFNNs in the modeling of SSME parameters. RBFNNs consist of a single layer of weighted, radially symmetric basis functions that are combined linearly. The parameters of an RBF network include the number, locations (centroids), and widths (variances) of the basis functions, and the amplitudes (weights) of these functions. The number, locations, and widths of the basis functions can be determined via a data clustering algorithm before the weight determination begins. RBFNNs are faster and easier to train than FFNNs, since the weights of the linear combiner are the only parameters to be learned during training. In general, RBFNNs are not as sensitive to architecture as BP feedforward networks.[10] Furthermore, RBFNNs find the global minimum and therefore have better performance in reproducibility of results.

In this paper, different architectures of RBFNNs have been implemented and compared for prediction of the high-pressure oxidizer turbine (HPOT) discharge temperature. This parameter was selected because it is redlined and because it exhibits a large nominal variation during the 6 s startup transient. SSME hotfire data were

Received Nov. 24, 1993; revision received Jan. 27, 1994; accepted for publication Jan. 31, 1994.

*Graduate Student, Electrical and Computer Engineering Department.
†Associate Professor, Electrical and Computer Engineering Department, Rhodes Hall, ML30.
‡Lewis Research Center Group.

used to train and validate the networks. The performance of the RBFNN is compared with that of a FFNN trained with the Quickprop learning algorithm.[11] Quickprop is a derivative of the backpropagation algorithm that typically exhibits faster convergence. In addition, the prediction capability of the trained RBFNN in response to simulated hard and linear-drift input failure conditions is presented.

Theory: RBFNNs for Function Approximation

RBFNNs have been shown to be universal function approximators[12,13] and have been related to interpolation theory.[14–17] The basic RBFNN shown in Fig. 1 consists of an input layer for input signal distribution, a single hidden layer of processing units, and an output summation unit.

An input vector x with components 1 to n is presented to each processing unit. A processing unit has a centroid vector c_i, which determines the location of the center of the radial basis function, and a variance σ_i^2, which determines the width of the function. The radial basis function is applied to the Euclidean distance between an input vector and its own centroid divided by the variance. The output of each unit is then weighted by w_i and summed by the output summation unit. Thus, the output $f(x)$ of the network is represented by:

$$f(x) = \sum_{i=1}^{K} w_i \phi(x, i) \tag{1}$$

where ϕ is the radial basis function (typically Gaussian)

$$\phi(x, i) = \exp\left(- \frac{\|x - c_i\|^2}{2\sigma_i^2} \right) \tag{2}$$

With this basic topology, the system can be represented in state-space form:

$$y = (F \cdot w) \tag{3}$$

where F is the matrix of activations:

$$F = \begin{pmatrix} \phi(x_1, 1) & \cdots & \phi(x_1, K) \\ \vdots & \ddots & \vdots \\ \phi(x_N, 1) & \cdots & \phi(x_N, K) \end{pmatrix} \tag{4}$$

The weights can be calculated using a pseudoinverse[18]:

$$w = (F^T \cdot F + \alpha I)^{-1} \cdot F^T \cdot y \tag{5}$$

where α is a constant less than 1, and I is the identity matrix. αI is added for two reasons: in the event that the square matrix $F^T \cdot F$ is close to being singular, and as a regularization term. In the event of noisy or redundant data, the square matrix $F^T \cdot F$ may be close to singular, which requires an additional term to obtain a numerical solution. When the weights of the network are solved via matrix inversion, the weight values are optimum in a least-squared-error sense, but the generalization capability of the network may not be optimal. The regularization term was therefore used to improve the generalization capability of the network, and to allow for the matrix inversion to be possible.

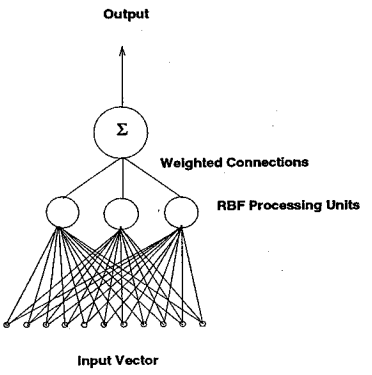


Fig. 1 Radial-basis-function neural network.

Internal normalization across Gaussian units can be implemented by adding lateral connections between the units. Without normalization, the output of each unit is calculated as in Eq. (2). The following equation incorporates normalization across all of the Gaussian units upon presentation of each input vector:

$$\frac{\phi(x, i)}{\sum_{i=1}^{K} \phi(x, i)} \tag{6}$$

The normalization forces the sum of the outputs of all units to be 1 for any input vector. This allows the output of smaller units to have a greater effect on the overall output. Thus, when an input vector falls between two units, the system may better interpolate.[10]

The parameters that need to be established are the number of centroids, their locations (means of the Gaussian units), and the widths (variances) of the radial basis functions. The locations and widths of the radial basis functions depend upon the data presented to the network. The K-means clustering algorithm can be used to determine the locations. The variances (widths of the RBFs) can be computed adaptively from the data using a neighborhood of p nearest vectors, where p is a predefined number.[19,20] Alternatively, the variance can be computed from the data using a neighborhood of p nearest centroids of surrounding clusters.[19,20] In the first method, the variance of a cluster is taken to be the mean of the squared Euclidean distances from the centroid of that cluster to its p nearest vectors in the training set. In the p-nearest-centroid method, the variance of a cluster is taken to be the mean of the squared Euclidean distances from the centroid of that cluster to the p nearest centroids of surrounding clusters.

RBFNN Methodology and Design Issues

The major design issues related to the RBFNN architectures are the number of RBF processing units or centroids, the appropriate regularization, and the parameters associated with each RBF processing unit. In addition, input parameters to the RBFNN model must be carefully selected to achieve good performance.

Since it was not practical to cover the entire input space uniformly with units, the K-means clustering algorithm was chosen to determine the centroids of the units. The K-means clustering algorithm requires some a priori information as to the number of clusters that should be formed. Ideally, the performance of the RBFNN should not be highly sensitive to the number of units selected. The goal was to provide both adequate generalization and accurate learning. There is a tradeoff between generalization and low error on the known data. Generalization can be made greater at the expense of increased error at the known data points. In order to address this issue, the number of centroids was varied from 10 to 100% of the original training data using the K-means clustering algorithm.

The appropriate regularization value depends strongly on the data. The performance of the RBFNN is compared with and without regularization; a suitable value was found via experimentation.

The variance associated with each centroid should reflect the distribution of the training data. Two methods to calculate the individual unit variance were investigated with the use of the regularization term. The first adaptive variance method was based on the mean of the squared Euclidean distances to the p nearest input data vectors. The second method was based on the mean of the squared Euclidean distances to the p nearest centroids.[19,20]

The choice of input parameters is critical in defining the accuracy of the neural model. There is a tradeoff between selecting a large enough number of inputs to allow sufficient generalization in the state space, and minimizing the number of inputs used to achieve efficient training. The set of inputs may be selected by using expert knowledge, or by an algorithmic approach such as a genetic algorithm described in Ref. 21. Results from Ref. 21 were used in this investigation. Since a neural network without feedback is used for time-series prediction, a time window about each input parameter was used to provide time-dependent information.

Application to SSME Data

The methodology presented in this paper could be used to model any SSME parameter that has sufficient analytical redundancy in the available instrumentation suite. The investigation presented here is limited to modeling the HPOT discharge temperature, a redlined parameter. This parameter is difficult to model because of limited instrumentation on the oxidizer side of the engine, and because of its large test-to-test variability. In this paper, both FFNNs and RBFNNs were used to develop models for the HPOT discharge temperature.

The sensors used as inputs to the FFNN and the RBFNN models are listed in Table 1. A window of the last five values in time was used for each of the five input parameters, thereby creating a 25-dimensional input vector. All data were scaled to a range of [−0.5, 0.5] prior to presentation to the network. The output of the network was scaled back to its original range before the error statistics were calculated. The statistics presented are the normalized root-mean-square error (NRMS) and the maximum percentage error. The NRMS error was calculated as follows:

$$\text{NRMS} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \hat{x}_i)^2}{\sum_{i=1}^{n} x_i^2}} \tag{7}$$

where n is the number of exemplars, $\hat{x}$ is the network's predicted output, and x is the actual output.

The startup data were sampled at 25 Hz from 0.4 to 5.96 s, resulting in 140 vectors per test firing. Four test firings (B1060, B1066, B1070, and B1077) were used for training. Seven test firings were used for validation: B1061, B1062, B1063, B1067, B1071, B1072, and B1075. The HPOT discharge temperature experiences a wide range of nominal behavior in both of these sets, as can be seen in Figs. 2 and 3.

The performance of the RBFNN is compared with that of an FFNN trained with Quickprop. The Quickprop learning algorithm[11] is an improvement over the traditional backpropagation learning algorithm.[9] The feedforward network had an input distribution layer of 25 units, a hidden layer of 10 units, and one output unit. This architecture was selected from a large number of implementations with variations in architecture such as the number of units in each layer and the number of layers. The best FFNN architecture was carefully selected from the analysis of error statistics.

The sensitivity of the RBFNN with a p-nearest-centroid adaptive variance was studied by simulating two different failures for the input fuel-preburner chamber pressure (FPB), in the B1061 training set. The first method involved a hard failure of 14.7 psia at engine start plus 3.0 s. The second method involved a drift fault, where the actual sensor value was forced to drift linearly by 100 psia/s, starting at engine start plus 2.0 s.

Table 1 Input-parameter description

Main-combustion-chamber oxidizer injection temperature
Fuel-preburner chamber pressure
High-pressure oxidizer pump inlet pressure
Low-presure oxidizer pump shaft speed
Oxidizer tank discharge temperature

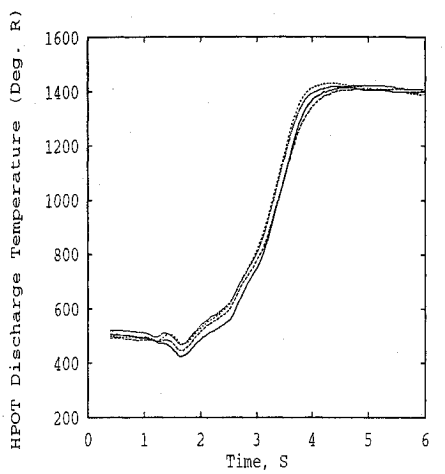


Fig. 2 Behavior of training sets: B1060, B1066, B1070, and B1077.

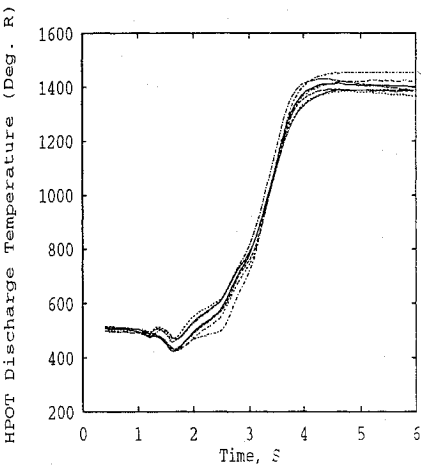


Fig. 3 Behavior of validation sets: B1061, B1062, B1063, B1067, B1071, B1072, B1075.

Results and Discussion

In order to determine the sensitivity of the RBFNN to variations in architecture, several experiments with different network parameters such as the number, location, and widths of the basis functions were conducted. The performance of RBFNN-based model was compared with that of the FFNN-based model. In addition, the performance of RBFNN was evaluated under simulated input-sensor failure conditions.

The locations of the RBFs were determined with the K-means clustering algorithm. Several networks were trained in order to study the sensitivity of the RBFNN's performance to the number of centroids. The K-means algorithm with K equal to 560, 280, 140, and 56 (100%, 50%, 25%, and 10% of the input training data) was used to form the K centroids of the network.

One network used an adaptive variance based on the 20 nearest input vectors with a regularization of 0.001. The neighborhood size and the regularization term were determined via experimentation. The error statistics for the training and validation sets for networks with 560, 280, 140, and 56 units are shown in Table 2. The table contains the NRMS error and the maximum percentage error. In comparing these tables, it can be seen that the statistical performance of the RBFNN was not significantly affected when a reasonable number of clusters was used. The network performance was not observed to be strongly tied to the number of nodes. From a computational standpoint, fewer nodes meant faster training and recall. When too few centroids were used, the network produced results that exhibited discrete steps; a slight deterioration in performance occurred with 56 nodes. It was observed that further reduction in the number of nodes contributed to larger errors. As a tradeoff between number of nodes and computation time, 140 nodes were chosen.

The regularization term, described previously, had the effect of smoothing the output of the network, unless the term was made too large. When the regularization term was too large, it caused the network's output to become oscillatory about the desired values. Based upon experimentation, the value chosen in this work was 0.001. Figure 4 shows the result of using a regularization of 0.001, and Fig. 5 shows the result with no regularization. The plots represent the response of the network when presented with validation set B1061. Tables 2 and 3 show the corresponding error statistics. Regularization provided a smoother output with lower maximum percentage errors on the validation firings.

In the p-nearest-vector-based adaptive variance method,[19,20] the variance of a cluster was assigned the mean of the squared Euclidean distance between its centroid and the p nearest input data vectors. One problem in choosing a fixed number of nearest input vectors as the neighborhood size is that there is no assurance there will be

Table 2 RBFNN error statistics with vector-based variance 20, regularization 0.001, for 560, 280, 140, and 56 units

Test firing	Train/ valid	560 units NRMS	560 units Max. % error	280 units NRMS	280 units Max. % error	140 units NRMS	140 units Max. % error	56 units NRMS	56 units Max. % error
B1060	T	0.017	7.371	0.017	7.473	0.018	7.754	0.020	7.680
B1066	T	0.012	3.295	0.011	3.220	0.013	3.738	0.017	6.583
B1070	T	0.018	5.060	0.017	4.924	0.017	4.858	0.021	6.862
B1077	T	0.009	3.968	0.009	4.011	0.011	4.728	0.014	5.385
B1061	V	0.013	6.885	0.013	6.858	0.013	7.045	0.012	7.410
B1062	V	0.016	6.515	0.016	6.528	0.018	6.533	0.019	6.974
B1063	V	0.024	4.889	0.024	5.115	0.024	4.876	0.026	5.525
B1067	V	0.022	4.702	0.022	4.814	0.023	4.959	0.023	5.600
B1071	V	0.034	10.806	0.034	10.959	0.035	10.930	0.038	14.830
B1072	V	0.023	4.298	0.022	4.202	0.024	4.901	0.025	6.754
B1075	V	0.023	8.490	0.023	8.352	0.024	9.107	0.022	8.687

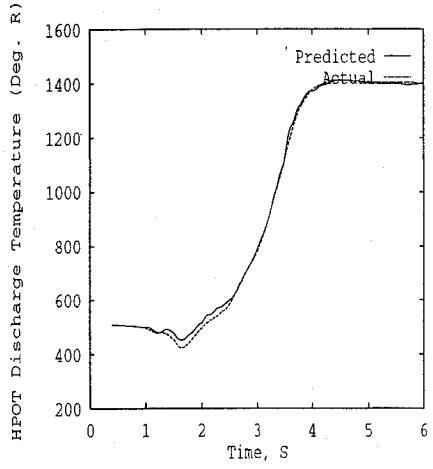


Fig. 4 Predicted and acutal values for a validation test firing B1061 for an RBFNN with 140 units, using 20-neighbor vector-based variance and regularization 0.001.

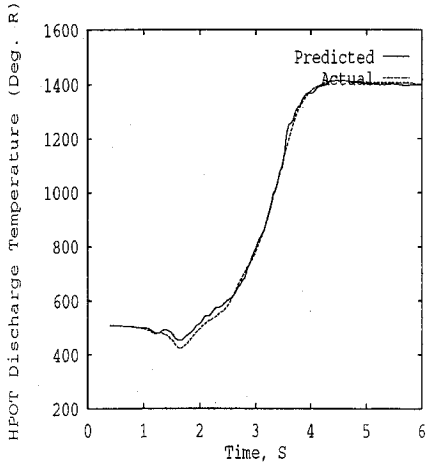


Fig. 5 Predicted and actual values for a validation test firing B1061 for an RBFNN with 140 units, using 20-neighbor vector-based variance, no regularization.

any overlap between adjacent clusters. Some overlap is desirable so that the transition from cluster to cluster will be smooth. If the number of clusters does not adequately represent the distribution of the input data, the variance computed from the nearest input vectors may introduce error into the computation of the RBF. When a cluster has a population that is small in comparison with the neighborhood size p, the variance of that cluster could be made artificially great and result in large error. When a cluster has a population that is large in comparison with p, the resulting variance may be made too small for adequate generalization.

Table 3 RBFNN error statistics with vector-based variance 20, no regularization, 140 units

Test firing	Train/ valid	NRMS	Max. % error
B1060	T	0.018	7.750
B1066	T	0.013	4.008
B1070	T	0.016	5.004
B1077	T	0.012	4.814
B1061	V	0.014	7.033
B1062	V	0.019	6.560
B1063	V	0.025	4.857
B1067	V	0.024	5.407
B1071	V	0.035	11.114
B1072	V	0.024	6.140
B1075	V	0.026	10.589

Since the K-means clustering algorithm formed clusters based upon the nonuniformly distributed data, the clusters did not have similar populations. This indicated the use of a centroid-based adaptive variance. This alternative adaptive-variance method used the p nearest centroids in the mean operation; the variance for a cluster became the average squared distance to the p nearest centroids. Because of the large variation in the data and the presence of noise, the results of the two adaptive variance methods were about the same.

The choice of neighborhood size is problem-dependent. A small value (2 to 10) proved adequate for the centroid-based adaptive variance method. A neighborhood of 2 produced results that were less smooth. Increasing the neighborhood size beyond 5 did not significantly improve the results. Therefore, a neighborhood size of 5 was selected. The error statistics and the output and error plots are shown in Table 4 and Fig. 6 respectively. In comparing the two RBF implementations with 140 units, Tables 2 and 4 show no significant differences in the error statistics.

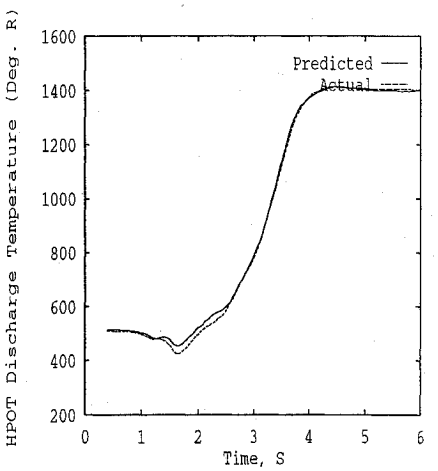
Several different Quickprop FFNNs were applied to the SSME data to determine the optimal architecture. The best architecture consisted of 25 units in the input layer, a hidden layer of 10 units, and 1 unit in the output layer. This architecture was determined after many trial-and-error attempts. The error statistics for the best configuration are shown in Table 4, and the corresponding output and error plots are shown in Fig. 7. Other configurations suffered from the problem of converging to higher error.

The best FFNN had error statistics and plots similar to the RBFNN shown in Table 4 and Fig. 6. The error statistics shown for the FFNN are for only one set of initial weights. However, these statistics are also representative of the average performance when different sets of randomly selected weights were used. In general, the performance of the two networks was comparable; however, the RBFNN model provided lower NRMS and percentage error on five out of seven of the validation firings.

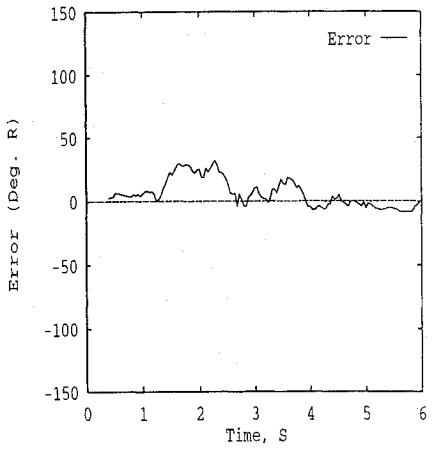
The FFNN took longer to train than the RBFNN. On average, the FFNN took 520 s, and the 140-unit RBFNN took 320 s to train on

Table 4 Error statistics for RBFNN with five-neighbor centroid-based variance, regularization 0.001, 140 units, and error statistics for a FFNN trained with Quickprop

		RBFNN		QP	
Test firing	Train / valid	NRMS	Max. % error	NRMS	Max. % error
B1060	T	0.018	7.912	0.021	8.638
B1066	T	0.014	3.719	0.017	4.212
B1070	T	0.018	5.041	0.023	6.351
B1077	T	0.010	4.365	0.014	4.594
B1061	V	0.012	7.060	0.014	7.474
B1062	V	0.017	6.402	0.022	7.694
B1063	V	0.025	5.431	0.025	5.192
B1067	V	0.023	4.567	0.025	5.248
B1071	V	0.035	12.903	0.042	14.838
B1072	V	0.024	4.451	0.028	4.883
B!075	V	0.022	8.087	0.021	8.972



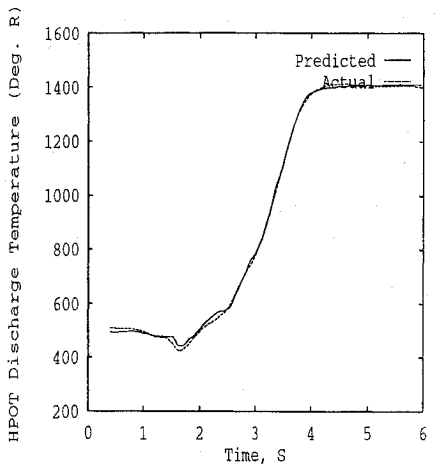
a) Predicted and actual values for a validation test firing B1061



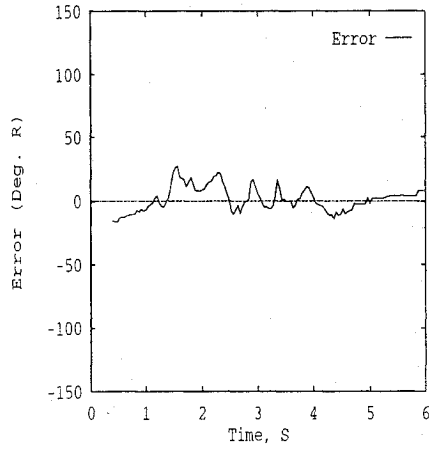
b) Corresponding error curve

Fig. 6 Performance of the RBFNN with 140 units, using 5-neighbor centroid-based variance, regularization 0.001.

a Sparc IPX running Sun 4.1.3 OS. The Quickprop algorithm used 321 parameters and trained for 2000 epochs, whereas the RBFNN used 420 parameters. One epoch represents one time through all of the training data. The Quickprop implementation used tables of discretized activation values rather than performing numerous sigmoid calculations. Our version of Quickprop was written in C++ and compiled with full optimization using the GNU Project C++ Compiler version 2.4. The RBFNN was written in C and compiled with full optimization using the GNU Project C Compiler version



a) Predicted and actual values for a validation test firing B1061



b) Corresponding error curve

Fig. 7 Performance of a feedforward neural network using the Quickprop algorithm.

2.4. Although the RBFNN used more parameters, it was not as sensitive to architecture, and did not suffer from the problem of falling into local minima on the error surface as did Quickprop. This is because the RBFNN consisted of only a single layer of linearly combined weights.

The generalization properties of the two types of networks are different. The RBFNN is a locally generalizing system, whereas the FFNN is a globally generalizing system. The effects of these characteristics can be seen in the plots. The Quickprop validation plot, Fig. 7, is smoother than the RBF validation plot in Fig. 6. By increasing the regularization of the RBFNN, it is possible to generate a smoother output at the cost of an overall increase in error.

The response of the trained RBFNN to simulated input sensor failures was also investigated. A 140-unit RBFNN, having a variance based on the five closest centroids and a regularization of 0.001, was trained on the original training set and then presented with faulted data. Both nominal and simulated fault values for FPB of test firing B1061 are shown in Figs. 8 and 9. Figure 8 shows the nominal and simulated hard-failure values for FPB, and the corresponding response of the trained RBFNN network. When a hard input failure was introduced, the network responded immediately. Because of the window size of five past values for each input, it took 0.2 s for the failed input to fill the window completely.

Figure 9 shows the nominal and simulated drift failure values for FPB, and the corresponding RBFNN network outputs. The simulated drift began at engine start plus 2.0 s. As the magnitude of the sensor failure increased, a significant deviation in the output value became evident. This shows that the performance of the RBFNN is robust to small deviations in the inputs.

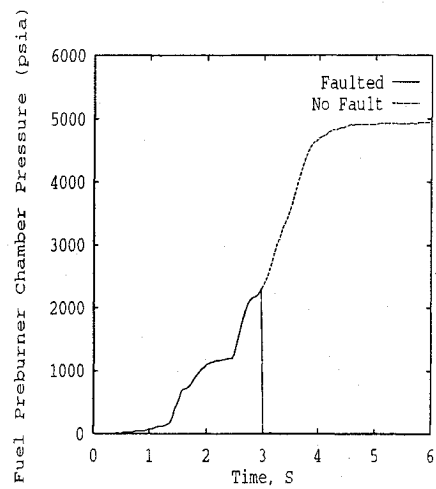


Fig. 8a Hard-faulted and nonfaulted input FPB.

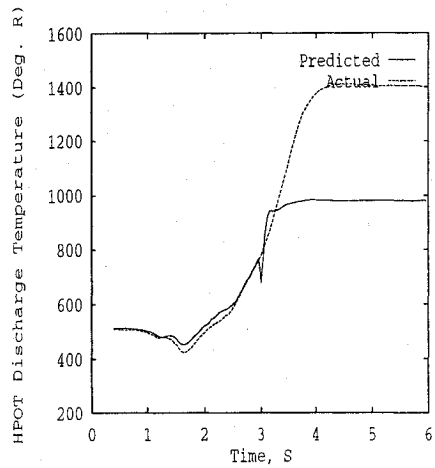


Fig. 8b Response of the RBFNN with 14 units, using 5-neighbor centroid-based variance and regularization 0.001, validation test firing B1061.

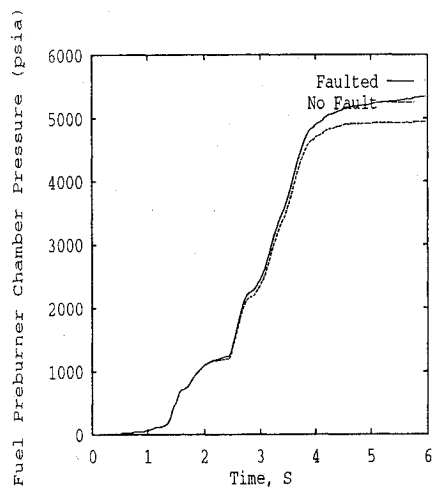


Fig. 9a Linear-drift-faulted and nonfaulted input FPB.

In addition to providing good generalization, the RBFNN provides a reasonable indication of both hard and drift input sensor failures. In an overall sensor validation system, the information from a large number of models would be combined to determine which sensors were faulty.

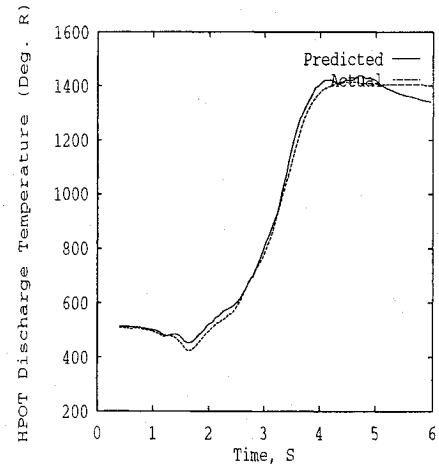


Fig. 9b Response of the RBFNN with 140 units, using 5-neighbor centroid-based variance and regularization 0.001, validation test firing B1061.

Concluding Remarks

The use of RBFNNs for modeling a critical Space Shuttle main-engine parameter has been investigated. The K-means clustering algorithm was used to determine the centroids of the Gaussian units. By having fewer radial-basis-function units than data samples, the weight training time was decreased. The performance of the RBFNN in predicting the high-pressure oxidizer turbine discharge temperature was compared with that of a feedforward neural network trained with Quickprop. The RBFNN was not sensitive to choices in the number of units or neighborhood size, as long as these choices were within a reasonable range. The feedforward neural network trained with Quickprop was sensitive to choices in architecture and learning rate.

The RBFNN had good generalization capability and provided good estimates of the high-pressure oxidizer turbine discharge temperature despite the widely varying behavior of the data. Equivalent performance was provided by adaptive variance methods based on the p nearest input vectors and on the p nearest centroids, but the latter variance is less sensitive to unequal cluster populations and neighborhood sizes. The best feedforward neural network and the RBFNN performed similarly with respect to error statistics, with lower error produced by the RBFNN. The RBFNN was able to train and recall faster than the feedforward neural network.

In the hard-failure input sensor condition, the RBFNN produced a faulted output. This is desirable in that the magnitude of the difference between the predicted and actual value of the HPOT will be indicative of a fault. For the linear-drift input failure condition, adequate generalization was available to produce a valid output within a small time after the occurrence of the fault. As the input deviation increased, the network produced an invalid output. This demonstrated that the RBFNN is robust to small deviations in the inputs.

The utility of the RBFNN approach could be further enhanced by automating the selection of the heuristic parameters, such as the number of units and the parameters involved in the variance calculation. Additional sensor models could be combined to form an automated sensor validation system.

Acknowledgments

This work was partially supported by grants from the NASA Lewis Research Center and from the NASA-UC Space Engineering Research Center for System Health Management Technology at the University of Cincinnati. The C++ Quickprop program used in this work was written by Charles Peck, by modifying Terry Regier's C implementation of Scott Fahlman's Quickprop training algorithm.[11] The pseudoinverse was performed using the iterated singular-value-decomposition matrix inversion routines found in Ref. 22.

References

[1]Bickmore, T., "Probabilistic Approach to Sensor Data Validation," AIAA Paper 92-3163, July 1992.

[2]Lin, C. S., Wu, I. C., and Guo, T. H., "Neural Networks for Sensor Failure Detection and Data Recovery," *Proceedings of International Conference on Artificial Neural Networks in Engineering*, St. Louis, Nov. 1991.

[3]Makel, D. K., Flaspohler, W. H., and Bickmore, T. W., "Sensor Data Validation and Reconstruction, Phase 1: System Architecture Study," NASA CR 187122, 1991.

[4]Meyer, C. M., and Maul, W. A., "The Application of Neural Networks to the SSME Startup Transient," AIAA Paper 91-2530, June 1991.

[5]Ruiz, C. A., and Hawman, M. W., and Galinaitis, W. S., "Algorithms for Real-Time Fault Detection of the Space Shuttle Main Engine," AIAA Paper 92-3167, July 1992.

[6]Cybenko, G., "Approximation by Superpositions of a Sigmoidal Function," *Mathematics of Control, Signals, and Systems*, Vol. 2, 1989, pp. 303–314.

[7]Hornik, K., "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks*, Vol. 2, 1989, pp. 359–366.

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

[9]Rumelhart, D. E., Hinton, G. E., and Williams, R. J., "Learning Internal Representations by Error Propagation," *Parallel Distributed Processing Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, edited D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, MIT Press, Cambridge, MA, 1986, pp. 318–364.

[10]Hertz, J., Krogh, A., and Palmer, R. G., *Introduction to the Theory of Neural Computation*, Addison-Wesley, Reading, MA, 1991, pp. 248–250.

[11]Fahlman, S. E., "An Empirical Study of Learning Speed in Back-Propagation Networks," Carnegie Mellow University, Technical Report CMU-CS-88-162, Pittsburgh, Sept. 1988.

[12]Park, J., and Sandberg, I. W., "Universal Approximation Using Radial-Basis-Function Networks," *Neural Computation*, Vol. 3, No. 2, 1991, pp. 246–257.

[13]Sun, X., "On the Solvability of Radial Function Interpolation," *Approximation Theory VI*, edited C. Chui, L. Schumaker, and J. Ward, Vol. 2, Academic Press, New York, 1989, pp. 643–646.

[14]Broomhead, D. S., and Lowe, D., "Multivariable Functional Interpolation and Adaptive Networks," *Complex Systems*, Vol. 2, 1988, pp. 321–355.

[15]Girosi, F., and Poggio, T., "Networks and the Best Approximation Property," MIT Artificial Intelligence Laboratory and Center for Biological Information Processing, Whitaker College, A. I. Memo 1164, C. B. I. P. No. 45, Oct. 1989.

[16]Lowe, D., "Adaptive Radial Basis Function Nonlinearities, and the Problem of Generalisation," *First Institute of Electrical and Electronics Engineers International Conference on Artificial Neural Networks*, Institute of Electrical and Electronics Engineers Conference Publication 313, London, Oct. 1989, pp. 171–175.

[17]Powell, M. J. D., "Radial Basis Functions for Multivariable Interpolation: A Review," *Algorithms for Approximation*, edited by J. C. Mason and M. G. Cox, Clarendon Press, Oxford, England, UK, 1987, pp. 143–167.

[18]Schalkoff, R. J., *Digital Image Processing and Computer Vision*, Wiley, New York, 1989, pp. 460.

[19]Moody, J., and Darken, C., "Learning with Localized Receptive Fields," *Proceedings of the 1988 Connectionist Models Summer School*, edited by D. Touretzky, G. Hinton, and T. Sejnowski, Morgan Kaufmann, San Mateo, CA, 1988, pp. 133–143.

[20]Moody, J., and Darken, C., "Fast Learning in Networks of Locally-Tuned Processing Units," *Neural Computation*, Vol. 1, No. 2, 1989, pp. 281–294.

[21]Peck, C., Dhawan, A., and Meyer, C., "Selection of Input Variables for SSME Parameter Modeling Using Genetic Algorithms and Neural Networks," *Proceedings of the Fourth Annual Space System Health Management Technology Conference*, Cincinnati, OH, 1992, pp. 104–118.

[22]Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T., *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge Univ. Press, New York, 1988, pp. 39–46.