EFFECTS OF LOCALIZED INPUTS ON NEURAL NETWORK TRAINING

Kristina A. Richardson * and Robert F. Stengel †
Department of Mechanical and Aerospace Engineering
Princeton University
Princeton, NJ 08544

ABSTRACT

Procedures for the design and training of a radial basis function network to represent dynamic aerodynamic data are outlined. The local support of the radial basis function network approximation allows new information from a dynamic maneuver to be considered, while retaining knowledge in regions not visited. The design procedure includes both the k-means clustering and extended Kalman filter algorithms for nodal parameter determination. The training procedure uses an input-space distance criterion to incorporate new information into the training set. The optimal number of network nodes is investigated through generalization ability. Numerical experimentation indicates that networks designed and trained with these algorithms learn locally and generalize to new data.

INTRODUCTION

Neural networks provide a powerful means of approximating multivariate, nonlinear functions, but learning during dynamic operation may be difficult due to the limited range of the training data. The state histories obtained during maneuvers are constrained by the system’s dynamic equations to follow restricted paths through the input space; thus the network input sample density is not uniform over the entire learning domain. The objective of the current work is to develop neural network design and training algorithms that effectively accommodate non-uniform data distributions.

Neural networks contain simple, nonlinear basis functions (or activation functions) connected with weighted links. The error between actual (or target) and approximating functions is minimized by proper choice of the network weights. Radial basis function (RBF) networks are attractive for function approximation during system operation because of the local support of the representation. A network must exhibit good generalization ability or the means to respond correctly to novel input data. Generalization ability varies with the ratio of the number of hidden nodes to the dimension of the training-set input vector.

Current research on adaptive system identification with RBF neural networks focuses on providing a local estimate of the function near an operating point rather than the construction of an accurate approximating function over the entire domain. These approaches take several different forms. A simple solution fixes the RBF centers and updates the output weights using a recursive least-squares algorithm. Adaptive RBF architectures (e.g., updates of output weights, centers and/or widths) were first suggested by Moody and Darken [1] and then applied to nonlinear system identification by Chen and Billings [2]. Since the entire architecture responds to the current training set, the performance of these algorithms can deteriorate when the operating condition changes [3]. Another approach is to use a growth criterion based on the increase or decrease of function approximation error by the addition or deletion of hidden units [4]. This approach does not necessarily optimize generalization ability or adapt well to changing approximating function shapes.

The RBF network structural design algorithm includes procedures for determining the nodal parameters and initializing the fixed-size training set D. Engineering estimates or model runs provide the design data. The nodal parameters include the function centers and widths and are chosen to satisfy a required sum-squared error ($J_{\text{design}}$). Initially, this is done with the k-means clustering algorithm [1, 5], and results indicate that the sum-squared error (SSE) decreases as the number of nodes is increased. If $J_{\text{design}}$ is not achieved with the k-means algorithm alone, the extended Kalman filter (EKF) algorithm [6, 7] is em-
ployed to further tune the center positions. The training set \( D \) is initialized with uniformly distributed design data, and entry of new data into \( D \) during training is controlled by satisfying an input-space distance criterion.

In this paper, RBF networks with varying numbers of hidden nodes are designed, and then trained with random and flight maneuver data on two different approximation surfaces. We begin with randomly distributed data drawn from the entire input space to learn a twin-jet transport model of the aerodynamic lift coefficient (Case I). A localized feature (or bump) is added to the lift coefficient surface, and the network is retrained with randomly distributed data (Case II). Finally, we use data drawn from a simulated flight maneuver to train the network to learn the "bump" surface (Case III). In all three situations, the generalization ability versus the number of hidden nodes in the networks is investigated, and a method for choosing the optimal number of hidden nodes is described. Finally, we examine the actual approximation performance of a network with the optimal number of hidden nodes during training.

### RADIAL BASIS FUNCTION NETWORKS

Consider a radial basis function network (Fig. 1) that maps a \( d \)-dimensional input space \( x \) to a one-dimensional target space \( y \). Data for parameter estimation includes \( N \) sets \( (x_n, y_n) \), where \( n = 1, \ldots, N \), including an input vector and its corresponding target. The network contains a set of \( m \) radial basis functions of the form \( \phi(x) = \phi(||x - c||) \), where \( x \) is the input vector, \( c \) is a fixed point referred to as the center of the function, and \( || \cdot || \) is the Euclidean norm. The links from \( x \) to the hidden nodes are unweighted. The weighted links from the hidden nodes to the linear output node are included in the \((m \times 1)\) vector \( w \). The network output is a scalar linear combination of the basis functions

\[
h(x) = \sum_{j=1}^{m} w_j \phi_j(||x - c||) \quad (1)
\]

The Gaussian function is a typical RBF:

\[
\phi_j(x) = \exp \left\{ -\frac{||x - c_j||^2}{2\sigma_j^2} \right\} \quad (2)
\]

where \( x \) is the input vector, \( c_j \) is a vector specifying the basis center of the \( j \)th basis function, and \( \sigma_j \) is the width. This can be generalized to allow for covariance matrices \( S_j \) and takes the form

\[
\phi_j(x) = \exp \left\{ -\frac{1}{2}(x - c_j)^T S_j^{-1}(x - c_j) \right\} \quad (3)
\]

Figure 1. Radial Basis Function Network.

The sum-squared error between the network and target outputs is written as

\[
SSE = \frac{1}{N} \sum_{n=1}^{N} (h(x_n) - y_n)^2 \quad (4)
\]

For \( m \leq N \), the solution for \( w \) that minimizes (4) is the least-squares solution.

If all \( N \) data sets are passed through the network simultaneously in a batch process, the network output is written in matrix form as

\[
h = \Phi w \quad (5)
\]

where \( \Phi \) is a \((N \times m)\) matrix of hidden-node outputs and \( h \) is the \((N \times 1)\) output vector.

### Unsupervised Versus Supervised Parameter Estimation

The RBF network nodal parameters are determined with supervised or unsupervised techniques. In supervised learning, the activation function centers and/or widths and output layer weights are determined simultaneously by nonlinear optimization [1]. Target data are considered; therefore, learning times are long and representations are broad. Unsupervised methods are faster than supervised techniques and provide localized approximations. Techniques such as clustering or maximum likelihood estimation place the centers to form a representation of the probability density of the input data. Widths are functions of the distance between centers. The output layer weights are found by a least-squares fit to the target data.

### Generalization Ability

The generalization ability of RBF networks varies with the ratio of the number of hidden nodes \( m \), to \( N \). Insight into generalization ability can be gained by
decomposing the generalization error into the sum of the square of the bias plus the variance [8]. Networks with too few nodes, or too little flexibility, have a high bias or average error. Too many nodes leads to high variance or sensitivity to particular data sets. For a given N, there is an optimal balance between bias and variance that yields the smallest average generalization error.

ALGORITHMS FOR NETWORK STRUCTURAL DESIGN AND TRAINING

The structural design algorithms use uniformly distributed data to set the activation function parameters and initialize D. The training algorithm considers new information obtained during dynamic system operation or simulation.

Network Structural Design Algorithms

Structural design algorithms include k-means clustering for determination of the function centers and Singular Value Decomposition (SVD) [9] for calculating the output layer weights. The function widths are set to one-half the Euclidean distance between the nearest centers. If necessary, the extended Kalman filter (EKF) is implemented for further minimization of the network output error.

k-means clustering - The k-means algorithm is unsupervised; it minimizes the sum of the squared distance between all the points in a cluster domain to the cluster center. The number of cluster centers, K, and points in the cluster domain, N, are fixed, and K < N. The cluster centers are held in a vector, c, and are initialized to random points in the cluster domain. The input data x are assigned to a cluster center according to

\[ x \in M_j(k) \text{ if } \| x - c_j(k) \| < \| x - c_i(k) \| \]  

for all i = 1, 2, ..., K, i ≠ j. M_j(k) denotes the set of samples whose cluster center is c_j(k), and k denotes the iterative step. After this assignment, new cluster centers c_j(k+1) are calculated according to

\[ c_j(k+1) = \frac{1}{N_j} \sum_{x \in M_j(k)} x, \ j = 1, 2, \ldots, K \]

where N_j is the total number of samples in cluster M_j(k). This two-step procedure is repeated until the cluster centers no longer move or c_j(k+1) = c_j(k) for j = 1, 2, ..., K is satisfied.

Determining Widths - Once the centers are fixed, the nodal widths are set at one-half the Euclidean distance between the nearest centers. These values are placed on the diagonal of the covariance matrix, S.

Singular Value Decomposition - The output layer weights of a RBF network are determined by minimizing the SSE defined in (4). This error function is quadratic in the weight vector, w, and its solution is determined by a set of linear equations.

If m = N and the inverse of \( \Phi \) exists, the solution for w is

\[ w = \Phi^{-1} y \]

For a large class of functions \( \phi(\cdot) \), including radial basis functions, the matrix \( \Phi \) is guaranteed to be non-singular provided the data points are distinct [10]. When the solution to w is given by (8), the network output h(x_n) is a continuous surface that passes exactly through each data point.

For m < n, the linear equations are

\[ \Phi^T \Phi w = \Phi^T y \]

and the solution is

\[ w = \Phi^T y \]

where \( \Phi^T = (\Phi^T \Phi)^{-1} \Phi^T \) denotes the left pseudoinverse of \( \Phi \).

The pseudoinverse calculation can have numerical difficulties if two basis vectors are nearly collinear (e.g., \( \Phi^T \Phi \) might be singular or nearly singular). Use of the SVD algorithm to calculate the inverse of \( \Phi \) alleviates numerical problems. The singular values \( \theta(\Phi) \) are the m positive square roots of the eigenvalues of \( \Phi^T \Phi \). In the SVD decomposition, \( \Phi \) is written as

\[ \Phi = UV^T \]

where U is a \( (m \times m) \) unitary matrix whose columns are the left singular vectors and \( V \) is a \( (N \times N) \) unitary matrix whose columns are the right singular vectors. The matrix \( L \) is a \( (m \times N) \) matrix with the m singular values in the diagonal of the leftmost \( m \times m \) portion of the matrix. The remaining \( (N - m) \) columns are filled with zeroes. The pseudoinverse of \( \Phi \) is now calculated by

\[ \Phi^T = VL^+U^T \]

where \( L^+ \) is a \( (N \times m) \) matrix which is the transpose of \( L \), with the singular values locally inverted.

Extended Kalman Filter with Additive Noise - The extended Kalman filter for neural networks is a
supervised algorithm that determines the network centers and output weights. These parameters are placed in the vector \( g \), where \( g^T = [w^T c_1^T \ldots c_n^T] \). This vector becomes the state of a nonlinear, discrete-time system. Equations for the propagation of the state and measurement vectors of this system are given by

\[
g_k = g_{k-1} + n_k \tag{13}
\]

\[
z_k = h(g_k) + v_k \tag{14}
\]

where \( n \in \mathbb{R}^n \), \( v \in \mathbb{R}^m \), and \( g() \) and \( h() \) are smooth nonlinear functions in \( \mathbb{R}^n \) and \( \mathbb{R}^m \) [11, 12]. The process and measurement noise sequences, \( n_k \) and \( v_k \), are zero-mean, white, and Gaussian, with covariances \( Q_k \) and \( R_k \). The general function \( h \) is defined by the network output, \( h = h(g) \). The initial state vector \( g_0 \) contains the centers and output weights from the appropriate structural design algorithm, and it has initial covariance \( P_0 \).

An extended Kalman filter for this system includes equations for covariance propagation, gain computation, and state and covariance update:

\[
P_k(-) = F_k P_{k-1} F_k^T + Q_{k-1} \tag{15}
\]

\[
K_k = P_k(\cdot)H_k^T[H_k P_k(\cdot)H_k^T + R_k]^{-1} \tag{16}
\]

\[
\dot{g}_k = \dot{g}_{k-1} + K_k [z_k - h(\hat{g}_{k-1})] \tag{17}
\]

\[
P_k = P_k(-) - K_k H_k P_k(\cdot) \tag{18}
\]

The symbol \( (\cdot) \) indicates an intermediate value after propagation but before updating with new information. The state Jacobian matrix \( F_k \) is the identity matrix for all \( k \), and the measurement Jacobian matrix is

\[
H_k = \frac{\partial h}{\partial g} |_{g_k} \tag{19}
\]

### AERODYNAMIC MODEL IDENTIFICATION

Networks were designed and trained for the identification of the lift coefficient of a twin-jet transport aircraft. Both uniform distribution and flight simulation training data were generated from a model of the aircraft. The model, the two multivariate training functions, and the training data sets are described below.

### The Lift Coefficient Model

The lift-coefficient training function is

\[
C_L(\alpha, q, \delta_E) = C_{L_{\alpha}}(\alpha) + C_{L_{q}} \frac{q^2}{2V} + C_{L_{\delta E}} \delta_E \tag{20}
\]

where \( \alpha \) is the angle of attack, \( q \) is the pitch rate, and \( \delta_E \) is the elevator deflection of the aircraft. \( V \) is the true airspeed, and \( \dot{c} \) is the mean aerodynamic chord of the wing. Values for the aerodynamic lift constants \( C_{L_{\alpha}}, C_{L_{q}}, C_{L_{\delta E}} \) are taken from [13]. The pitch rate derivative \( (C_{L_q}) \) is 7.0, and the elevator angle derivative \( (C_{L_{\delta E}}) \) is 0.006 deg\(^{-1}\). \( C_{L_{\alpha}}(\alpha) \) is a linear function of \( \alpha \) at low angles and becomes a quadratic function at higher angles according to

\[
C_{L_{\alpha}}(\alpha) = 0.09550\alpha + 0.1048, \quad \alpha < 9 \text{ deg}
\]

\[
C_{L_{\alpha}}(\alpha) = -0.0955\alpha^2 + 0.2667\alpha - 0.6667, \quad \alpha \geq 9 \text{ deg} \tag{21}
\]

The above model is valid for input data bounded by

\[
\alpha \in [-5 \text{ deg}, 19 \text{ deg}]
\]

\[
q \in [-20 \text{ deg/s}, 20 \text{ deg/s}]
\]

\[
\delta_E \in [-20 \text{ deg}, 20 \text{ deg}].
\]

### Two Multivariate Training Functions

The “original” lift coefficient surface (Fig. 2), considered during Case I, is generated by varying the pitch rate, angle of attack, and elevator deflection through their valid ranges. The pitch-rate effect is seen in the thickness of the \( C_L \) surface.

A bump training surface (Fig. 3) was generated from a modified version of the model above. This target includes a bump added to the \( C_L \) surface defined as

\[
C_L(\alpha, q, \delta_E) = C_L f_1 f_2 \tag{22}
\]

where

\[
f_1 = 0.0029 (\alpha - 3.5)^2 - 2
\]

\[
f_2 = 0.007 (\delta_E - 3.5)^2 - 5 \tag{23}
\]

in
Figure 2. Original Model Relationship Between the Lift Coefficient, Angle of Attack, and Elevator Deflection - Used to Design and Train the Network (Case I) - Representative Random Uniform Distribution $\alpha$ and $\delta_E$ Training Data (Cases II and III).

$$\alpha \in [1.1 \text{ deg}, 13.8 \text{ deg}]$$  
$$q \in [-20 \text{ deg/s}, 20 \text{ deg/s}]$$  
$$\delta_E \in [-7 \text{ deg}, 14 \text{ deg}]$$

and $C_L$ is calculated from eq. 20. This surface is considered to test the malleability of the network approximation or its ability to change shape and adjust to the actual coefficient surface during flight (Cases II and III).

Angle of attack, elevator deflection, and pitch rate test sets for both surfaces are included in Fig. 4. The box in Fig. 4 surrounds input data sets whose target data are affected by the bump.

**Data for Structural Design and Training**

Structural Design Data - Structural design data are drawn from the original coefficient surface discussed above (Fig. 2), with 1000 training sets that cover the input space. Angle of attack, elevator deflection, and pitch rate design input data are represented by the markers in Fig. 4.

Training Data - Both the original and bump surfaces provide training data for the networks. First, the training algorithm is provided random data from a uniform distribution drawn from the original surface (Case I). Next, random data are drawn from the bump surface (Case II). Representative random uniform distribution training data for both these cases are indicated by the markers on the $\alpha$-$\delta_E$ plane in Fig. 2.

Finally, a simulated 3-2-1-1 flight produces state histories through the input space (Case III). Target training data are drawn from the bump coefficient surface. The 3-2-1-1 maneuver is used due to the number of nodes it excites and the ease of its execution [14]. "3-2-1-1" refers to the time durations of an alternating control input (i.e., elevator deflection). The $\delta_E$ (Fig. 5). The $\alpha$ and $\delta_E$ input training data produced by a 3-2-1-1 flight maneuver are indicated by markers on the $\alpha$-$\delta_E$ plane in Fig. 3.

**DISCUSSION**

The optimal number (or range) of RBF network hidden nodes for training is investigated from the viewpoint of generalization ability. Random uniform distribution and flight simulation training data and the original and bump surfaces are considered. Once the number of network hidden nodes is fixed,
the training algorithm's performance is demonstrated during a 3-2-1-1 flight maneuver on the bump multivariate function.

Network Structural Design

Generalization ability is evaluated by varying the ratio of the number of hidden nodes to the size of the training set (which is fixed). For each ratio value, the performance is evaluated over a set of networks. The ratio is incremented from 0.05 to 0.28, and 80 networks are designed for each value. To begin the design procedure, \( D \) is initialized with 1000 training pairs \([x_n, y_n]\) drawn from a uniform distribution. The required SSE between the network and target outputs before flight simulation training can begin is \( 9 \times 10^{-6} \) (i.e., \( J_{\text{design}} = \sqrt{\text{SSE}} = 0.003 \)). In networks with 100 or more nodes, use of the \( k \)-means algorithm (e.g., \( k \)-means determined the function centers, widths were set to one-half the Euclidean distance between the nearest centers, SVD determines the output weights) provides parameter values that yield SSE's less than \( J_{\text{design}} \).

A combination of the \( k \)-means and EKF algorithms determined the parameters in networks with less than 100 nodes. The state vector \( g_0 \) is initialized with the center values from the \( k \)-means algorithm and weight values from the SVD algorithm. These values were then tuned with the EKF algorithm until a performance metric less than \( J_{\text{design}} \) was attained. Observation of the final positions of the centers indicates that a small percentage (e.g., < 5%) of the centers moved outside the input-space bounds. The initial value of the state covariance matrix \( P_0 \) is \( 10I_n \), the measurement noise covariance matrix \( R \) is \( 10^{-1}I_m \), and the process noise covariance matrix \( Q \) is \( 10I_n \).

Generalization Ability of Trained Networks - Random Training Data (Cases I and II)

Three-Input Network - After a network is designed, training begins, with data in the training set \( D \) replaced when an input-space distance criterion is satisfied. First, training pairs are randomly drawn from the original surface (Case I). The training data included 3,333 sets. In order to consider generalization ability of the final trained network, the bias, standard deviation, and \( J_{\text{train-gen}} (J_{RG}) \) on a test data set, a set of 1,000 data sets that the network has not seen during design or training, were evaluated. Following this convention, the performance metric, \( J_{RG} \), is the SSE of the trained network on the test data. It provides a measure of the network's ability to generalize to new data. Figure 6 includes these values as the total number of hidden nodes is varied from 50 to 280 and the networks attempt to learn the original multivariate function. Figure 7 depicts performance when the networks attempt to learn the bump function with random training data (Case II).

In both figures, the bias decreases and the variance increases as the number of hidden nodes increases, as expected. As the number of hidden nodes is increased, the network output becomes closer to the training data, and the output of each network is closer to its particular training data. On the original function, the minimum \( J_{RG} \) is achieved with 160 hidden nodes, while in the bump case, the minimum \( J_{RG} \) is achieved with 220 hidden nodes. The curves in Fig. 7 indicate that networks with 50 to 120 nodes have difficulty discerning the bump in the surface. These results are intuitively appealing because one would expect to need more nodes to accomplish the task.

Two-Input Network - Two-input neural networks also were investigated; learning the same lift coefficient surface but with \( a_n \) and \( \delta_e \) varying and \( q \) kept constant at 0 deg/s. The training set, \( D \), includes 1,000 training points, 100 test points, and the number of hidden nodes is varied from 15 to 90. Best generalization performance with random uniform distribution training data on the original surface was observed with 15 nodes. The minimum \( J_{RG} \) occurred at 40 nodes. These results displayed the same trends as seen above – more nodes were required to discern a bump in the surface. Comparison of the three-input and two-input results revealed a curse of dimensionality. In the three-input case, the test set size increased ten-fold (as expected), but the growth in the required number of hidden nodes to over 100 seems high considering the shape of the curve has essentially remained the same. For higher input problems,
this trend would lead to computationally prohibitive situations.

Generalization Ability of Trained Networks - 3-2-1-1 Flight Maneuver Training Data (Case III)

Three-Input Network - Bias, standard deviation, and $J_{TG}$ were calculated on network sets trained with 3-2-1-1 flight maneuver training data. Figure 8 includes the bias, standard deviation, and $J_{TG}$ as the total number of hidden nodes is varied from 50 to 280 and the networks attempt to learn the bump multivariate function (Case III). Once again, the bias decreases and the variance increases as the number of hidden nodes is increased. The minimum $J_{train}$ ($J_T$) is achieved with 100 hidden nodes, but the performance metric is within 5% of the minimum value for hidden nodes in the range of 50-180 nodes. The performance metrics for 3-2-1-1 training data (Fig. 8) are higher than metrics achieved with random training data (Fig. 7). This difference is due to the smaller training region in the 3-2-1-1 case; only 4% of the entire input space is encountered during this simple maneuver. Performance metrics are higher because less of $D$ has been modified. The basic nature or shape of these curves is different because different functions are being approximated. In Fig. 7 the entire bump surface is approximated; the Fig. 8 approximating function is defined by the data sets left over from the initialization of $D$ and those obtained during 3-2-1-1 training.

In order to better understand Fig. 8, Figure 9 includes $J_{DG}$ from Fig. 8 decomposed into four performance metrics; $J_{design-gen-bump}$ ($J_{DGB}$) and $J_{TGB}$ on the 200 test points associated with the bump portion of the surface and $J_{design-gen-outside}$ ($J_{DGO}$) and $J_{TGO}$ over the rest of the test points. After design, the performance metrics are flat over both the bump and the outside portions. After training or flight, the performance metric on the bump portion of the surface remained flat and decreased about 0.001 in magnitude; a small, but reasonable amount because only a small portion of the bump surface, with respect to $q$, is encountered during the maneuver. To better demonstrate how little of the bump was encountered during training, Fig. 10, displays
the first 100 seconds of the evolution of $q$ during the maneuver. The maximum value of $q$ was 13 deg/sec and the minimum was -5 deg/sec; in actuality, the bump extends for $q$ values from -20 to 20 deg/sec. Also, the circle markers in Fig. 10 denote values of $q$ where the combination of $[\alpha, \delta_E, q]$ combine to produce a training point in the bump region. With respect to $\alpha$ and $\delta_E$, the bump is covered well (see Fig. 3).

The $J_{TOO}$ curve in Fig. 9 explains the rise in $J_{TO}$ in Fig. 8 as the number of nodes increases; with more than 190 nodes the network start to unlearn the non-bump portion of the surface. There is numerical ill-conditioning because centers are now closer together, some basis function are nearly collinear, causes the non-bump surface performance metrics to increase.

Supervised Training - Supervised training algorithms by themselves also were examined; observations on the approximating behavior of these networks are worth mentioning. The training algorithm was the extended Kalman filter with the network centers and output weights in the state vector. The state vector was initialized with random values; centers within the input-space bounds and weights between -1 and 1. Design training times were up to an hour longer with the supervised algorithm than with the unsupervised algorithms above. With a random initial state vector, final design center positions were well outside the input-space bounds; thus the activation functions were broad and not local after design. During training, the EKF algorithm sequentially processed each training data set $(x_n, y_n)$ so that the state update eqn. (17) became

$$\hat{x}_k = \hat{x}_{k-1} + K_k (z_k - h[\hat{x}_{k-1}]) \quad (24)$$

where the measurement and network output vectors become scalars. The broad representations provided by the new design algorithm lead to several nodes being activated by each training data set; thus the network unlearned information about regions outside the input/output space encountered during training.

Network Performance with the Optimal Number of Nodes on Simulated Flight Data

Once generalization ability is used to select the optimal number of nodes, the approximation behavior of a network with 160 nodes learning the bump surface with flight simulation data can be examined. The training data includes an additional 100 time steps after the initial trajectory at a constant $\alpha$, $\delta_E$, and $q$ position. Figure 11 includes the evolution of the total performance metric, $J_T$, the bump $J_{TB}$, and the outside performance metric $J_{TO}$ versus time steps as the 3-2-1-1 flight maneuver is undertaken. The initial performance metric on the bump surface is 0.006, after design. The approximation error does not increase or unlearn in areas not visited. The performance metric $J_{TB}$ is 0.005 at the end of the training run.

CONCLUSIONS

The design and training algorithms outlined in this paper combine to yield network architectures that are able to learn multivariate functions with flight simulation data. The unsupervised $k$-means design algorithm is fast compared to supervised batch techniques and yields adequate SSE architectures when enough
hidden nodes are included. Application of the EKF algorithm to networks with lower numbers of hidden nodes allows us to consider these architectures because a minimum error metric requirement is then satisfied. Considerable insight into the performance of both the design and training algorithms was gained from the investigation of generalization ability. Results indicate that higher numbers of hidden nodes are required to discern bumps in surfaces. While the algorithm suffers from the curse of dimensionality with respect to the number of inputs, numerical experimentation indicates that networks designed and trained with these algorithms learn locally and generalize to new data in low-dimension input problems.

References


