01 Jan 2003

Query-Based Learning for Aerospace Applications

Donald C. Wunsch
Missouri University of Science and Technology, dwunsch@mst.edu

Emad W. Saad

J. J. Choi

J. L. Vian

Recommended Citation

D. C. Wunsch et al., "Query-Based Learning for Aerospace Applications," *IEEE Transactions on Neural Networks*, Institute of Electrical and Electronics Engineers (IEEE), Jan 2003.
The definitive version is available at https://doi.org/10.1109/TNN.2003.820826

Query-Based Learning for Aerospace Applications

Emad W. Saad, Jai J. Choi, John L. Vian, and Donald C. Wunsch, II

***Abstract*—Models of real-world applications often include a large number of parameters with a wide dynamic range, which contributes to the difficulties of neural network training. Creating the training data set for such applications becomes costly, if not impossible. In order to overcome the challenge, one can employ an active learning technique known as Query-Based Learning (QBL) to add performance-critical data to the training set during the learning phase, thereby efficiently improving the overall learning/generalization. The performance-critical data can be obtained using an inverse mapping called network inversion (discrete network inversion and continuous network inversion) followed by oracle query. This paper investigates the use of both inversion techniques for QBL learning, and introduces an original heuristic to select the inversion target values for continuous network inversion method. Efficiency and generalization was further enhanced by employing node decoupled extended Kalman filter (NDEKF) training and a causality index (CI) as a means to reduce the input search dimensionality. The benefits of the overall QBL approach are experimentally demonstrated in two aerospace applications: a classification problem with large input space and a control distribution problem.**

***Index Terms*—Active learning, aerospace application, causality index (CI), control distribution, crew escape system, efficient training, emergency egress safety, mapping, network inversion, node decoupled extended Kalman filter (NDEKF), pattern recognition, query-based learning (QBL).**

I. Introduction

THE quality of the training data is a key ingredient for neural network (NN) application. In aerospace industry, data generation sometimes can be very expensive, if not impossible. This reality motivates us not only to seek economical data generation techniques but also to investigate training methods that can utilize a limited amount of data. A supervised learning technique with a teacher (oracle) in the loop, known as query-based learning (QBL) [1]–[4] has been proposed to address the practical needs by incrementally adding training data as the learning progresses. QBL is an active learning technique [5], [6] where the learner actively selects its training data, as opposed to passive learning, where the learner uses a fixed set of training data repeatedly. For instance, optimal experiment design, [7], [8] actively selects the new training pattern in every learning cycle to minimize the learner's variance. Sample query [9] method is an incremental approach, which adaptively changes the sample size taken from each class. In pedagogical pattern selection approach [10], the training patterns are presented when specific learning error conditions are met. These active learning techniques are more complex and expensive due to the additional data generation processes [5], [11]–[13].

Most of these methods are devoted to pattern classification problems. When QBL is applied to a classification problem [1], a typical scenario is to train a NN using a small number of training data. In order to assess the learning status, one can invert the network using the network inversion method described below. Once created, each inversion vector is then queried through an oracle to find the corresponding target. Recall that, in supervised learning, the training data consists of input and target values. The newly created training data are then added to the existing data set in order to continue training. Because of the interaction by the query mechanism, the overall learning is called QBL.

The QBL process hinges upon network inversion. The network inversion iteratively searches for vectors, in the input space, called inversion vectors, which give rise to a specific inversion target value.[1] The network weight values need to be fixed for the network inversion. We can categorize the inversion into two types: discrete network inversion and continuous network inversion. Each method depends on the types of output values—the discrete value case, which is typical in classification problems, and the continuous case typical in continuous mapping/prediction problems. Accordingly, we use the term discrete network inversion and continuous network inversion to reflect the nature of the NN output.

The selection of the inversion target value is relatively intuitive in discrete network inversion. For example, in a binary classification problem (two classes that use 0 and 1 as network target value), the inversion target of $t = 0.5$ represents the classification boundary. The inversion vectors obtained above can be interpreted as the network version of the classification boundary. We can use these vectors to gauge the status of learning. Obviously, the closer the inversion vectors to the true classification boundary, the better the network performance becomes. One can also invert the partially trained network for a range of target values $[0.4 < t < 0.6]$ for instance, instead of a single target inversion strategy. For continuous network inversion, the choice of the inversion target value is not trivial. One needs to know what range of target values is worthy of network inversion. One way to find the critical value (or range of values) is based on the absolute training error. We can apply inversion whenever the network produces a large error that is outside of a preset threshold. In that case, the output value of the network can be directly used as the inversion target value.

[1]Note that this value is different from the target value of training data.

Manuscript received May 15, 2000; revised May 30, 2001 and March 3, 2003.

E. W. Saad is with Southern Methodist University, Richardson, TX 75081 USA (e-mail: esaad@engr.smu.edu).

J. J. Choi is with Boeing Phantom Works, Seattle, WA98124 USA (e-mail: jai.j.choi@boeing.com). He is also with the University of Washington, Seattle, WA 98195 USA.

J. L. Vian is with Boeing Phantom Works, Seattle, WA 98124 USA (e-mail: john.vian@boeing.com).

D. C. Wunsch, II is with University of Missouri-Rolla, Rolla, MO 65409 USA (e-mail: dwunsch@ece.umr.edu).

Digital Object Identifier 10.1109/TNN.2003.820826

In order for an inversion vector to be useful for training, we need to obtain the true target value of the vector by querying it to an oracle. An oracle (teacher) exists in a variety of forms such as mathematical equations, simulation models, human experts, or experiments.

In this paper, for the sake of notations and completeness, we review a formulation of discrete network inversion followed by QBL learning process in Section II. A technique of generating continuous network inversion data is also presented. In Section III and IV, applications of the QBL methods are detailed for two aerospace problems, ejection safety classification and ejection seat modeling, respectively. The former addresses decision on the ejection safety level based on the initial state of ejection parameters of a fighter jet ejection seat. This problem is characterized by a huge search space due to the high dimensionality and wide dynamic range of input parameters. The second application is a continuous mapping problem that maps propulsion control moments to nozzle commands in a four-nozzle controllable ejection seat. In this application, it is desirable to reduce the maximum absolute value of the network error. Based on the simulation studies through these applications, we draw some conclusions in Section V.

II. QBL

A. Discrete Network Inversion

What the trained network perceives about the classification boundary may be totally different from the true boundary. This is bound to happen when we have inadequate training data or in the early stage of network learning. If there is a way to ask the trained network about what it does or does not understand about the input data, that would increase the chance to improve on-going training. The network inversion [14] precisely addresses this issue. In order to establish notations, we first lay out the basic network structure. During the forward path of network training, the activation output of a neuron unit is given as

$$u_i = \sum_{j=0}^{p} w_{ij} x_j \tag{1}$$

and

$$x_j = f(u_j) \tag{2}$$

where p is the number of neurons in the previous layer, w_{i0} is the threshold, $f(.)$ is an activation function, and x represents the output of a unit. The most commonly used activation functions are the sigmoid

$$f(x) = \frac{1}{1+e^{-x}} \tag{3}$$

or the hyperbolic tangent

$$f(x) = \tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} = \frac{1-e^{-2x}}{1+e^{-2x}}. \tag{4}$$

The latter is asymmetric, and has the advantage of accelerating the learning process [15]. In the activation functions (3) and (4), a scaling factor b can be introduced for better learning [15] such that (3) becomes

$$f(x) = \frac{1}{1+e^{-bx}} \tag{5}$$

and (4) becomes

$$f(x) = \tanh(bx) = \frac{e^{bx}-e^{-bx}}{e^{bx}+e^{-bx}} = \frac{1-e^{-2bx}}{1+e^{-2bx}}. \tag{6}$$

The forward mapping from input to output is achieved by finding a set of weights which minimizes the sum squared error

$$E' = \frac{1}{2}\sum_{k=1}^{N_L} (t'_k - y_k)^2, \tag{7}$$

where t'_k is the target output, y_k is the actual network output, and N_L is the number of neurons in the output layer.

The reverse process of producing input vectors that yield a predetermined target output $\widehat{t}_k$ is referred to as network inversion [14]. The inversion target output is usually chosen to represent the output values which are hard for the network to learn. For example, in case of a neural network used for classification with binary output (0 and 1), the inversion target output value would be 0.5 in order to represent the decision boundary. For a given inversion target value, $\widehat{t}_k$, the corresponding input vectors of length m, $\vec{x} = (x_1^0, x_2^0, \cdots, x_i^0, \cdots, x_m^0)$, can be obtained by back-propagating the inversion error, E, given in (8), through a partially trained network with it's weight values frozen.

$$E = \frac{1}{2}\sum_{k=1}^{N_L} (\widehat{t}_k - y_k)^2. \tag{8}$$

The inversion process is an iterative search method in the input space. Once an initial search point with components $u_i^0(0)$ is randomly assigned, the update rule for the activation potential u_i^0 in the input layer at the $(t+1)^{\text{th}}$ iteration is given as

$$\begin{aligned} u_i^0(t+1) &= u_i^0(t) - \eta \frac{\partial E}{\partial u_i^0(t)} \\ &= u_i^0(t) - \eta \frac{\partial E}{\partial x_i^0(t)} \frac{\partial x_i^0(t)}{\partial u_i^0(t)} \\ &= u_i^0(t) - \eta \delta_i^0(t) \frac{\partial x_i^0(t)}{\partial u_i^0(t)}. \end{aligned} \tag{9}$$

The derivative $\delta_i^l(t)$ for the neuron units in layer l is obtained by a chain rule [16] as follows:

$$\begin{aligned} \delta_i^l &= \frac{\partial E}{\partial x_i^l} \\ &= \sum_{j=1}^{N_{l+1}} \frac{\partial E}{\partial x_j^{l+1}} \frac{\partial x_j^{l+1}}{\partial x_i^l} \\ &= \sum_{j=1}^{N_{l+1}} \delta_j^{l+1} w_{ji}^{l+1} \frac{\partial x_j^{l+1}}{\partial u_j^{l+1}}. \end{aligned} \tag{10}$$

For the neurons in the output layer, the derivative becomes

$$\begin{aligned} \delta_i^L &= -\left(\widehat{t}_i - x_i^L\right) \\ &= -(\widehat{t}_i - y_i). \end{aligned} \tag{11}$$

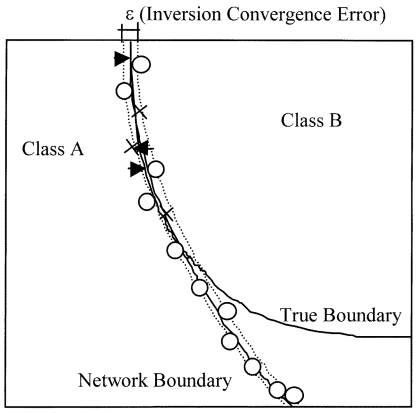


Fig. 1. Effect of adding jitters to query data. The points closer to the true boundary may move to the opposite class due to the added jitter, while the points far away from the decision boundary remain unaffected. Points from class A are shown as o's, while points from class B are shown as x's. The arrows indicate the direction in which a point was moved by jitter and crossed the boundary.

The inversion vector components at the input layer is obtained by

$$x_i^0 = f\left(u_i^0\right) \tag{12}$$

which guarantees that the network input will be in the range of the activation function. Thus, constrained iterative inversion is used as explained in [14].

The inversion vectors are then queried to an oracle $\psi(\cdot)$ to produce the true target values $t' = \psi(\vec{x})$ for a newly generated set of training data $(\vec{x}, t')$. We can also add random noise (jitter) $\vec{n}$ to produce another training vector $(\vec{x} + \vec{n}, t')$.

While adding jitter (noise) to the input data without correcting the target output might seem to adversely affect the network training, it has been shown that the strategy has the effect of smoothing the decision boundary and actually improving the generalization [17]–[26]. Depending on the variance of the noise, jitter may have a greater impact on the data points closer to the decision boundary as illustrated in Fig. 1. These points, when jittered, can move across the decision boundary. On the other hand, the data points farther away from the decision boundary are not affected by adding jitters. Can this be the case for query based learning, especially when the number of data points along the boundary is sparse? In this case, we believe that when the decision boundary is smooth and the classes are well separated, adding jitter may adversely impact the learning. We will investigate this conjecture in our experiments in a later section.

B. Continuous Network Inversion

As we mentioned in an earlier section, the choice of the inversion target value for continuous network inversion is a challenging task. Very limited work has been devoted in this area (in [27] an active learning is applied to locally weighted regression). In the control distribution application, we are most interested in minimizing the maximum error at the output, not just the rms error. In order to reduce such errors, we pre-set an acceptable error bound (threshold) and use it as a guide for the continuous network inversion.

The proposed continuous network inversion is as follows: Upon freezing the network weights, a test data set is fed to the trained network. The test data is generated in a fashion similar to the training data using the oracle and is uniformly distributed over the input space. The cost of the test data generation is equal to the cost of the training data generation, but is obviously generated only once and does not need inversion. The corresponding network output is then compared with the desired target value. Whenever the absolute error $(= |\text{target value} - \text{actual output value}|)$ is greater than a pre-set threshold (the choice of which relies on personal discretion), the actual neural network output value is used as the inversion target value. Once, of course, having set the inversion target value, the inversion algorithm is identical to the discrete network inversion case. During the inversion process, the network input is initialized using a random value as explained in Section II-A. Therefore the network output is no more equal to the target value. The nonzero error gradient is used to change the input value until another input point is found where the output error is close to zero. Iteratively, using different initial random input points, several input patterns are obtained, for each inversion target value.

C. Practical Techniques for Efficient Training: Causality Index and Error Bias

1) Causality Index: The causality index (CI) as a means to measure the output sensitivity to the network input, is used to improve the generalization capability [29], [30]. Based on the dependency of an output on each input, we can eliminate certain input parameters with low sensitivity. Obviously the larger the number of inputs eliminated based on CI analysis, the faster the network inversion becomes. Consequently, it speeds up the QBL process. Note that the network inversion is a search algorithm in the input space based on the error observation in the network output. The CI value for a feed-forward network with a single hidden layer having N_1 hidden units is computed by

$$CI_{ki} = \sum_{j=1}^{N_1} w_{kj} \cdot w_{ji} \tag{13}$$

where w_{kj} is the weight value between the kth output unit and the jth unit in the hidden layer. w_{ji} is the weight value between the jth hidden unit and the ith input unit.

In order to compare CI values of different NN architectures, a normalized causality index is defined as

$$CI'_{ki} = \frac{\sum_{j=1}^{N_1} w_{kj} \cdot w_{ji}}{\sqrt{\frac{1}{N_L N_0} \sum_{m=1}^{N_L} \sum_{l=1}^{N_0} \left(\sum_{j=1}^{N_1} w_{mj} \cdot w_{jl}\right)^2}} \tag{14}$$

where N_L and N_0 are the number of output and input neurons, respectively. The causality index provides the following practical insights into feed-forward NN applications.

1) The causality index defined in (13) and (14) measures the average sensitivity of the output node k with respect to the input node i.
2) A positive causality index means that an increase in the input causes an increase in the output and vice versa. A negative causality index means that an increase in the

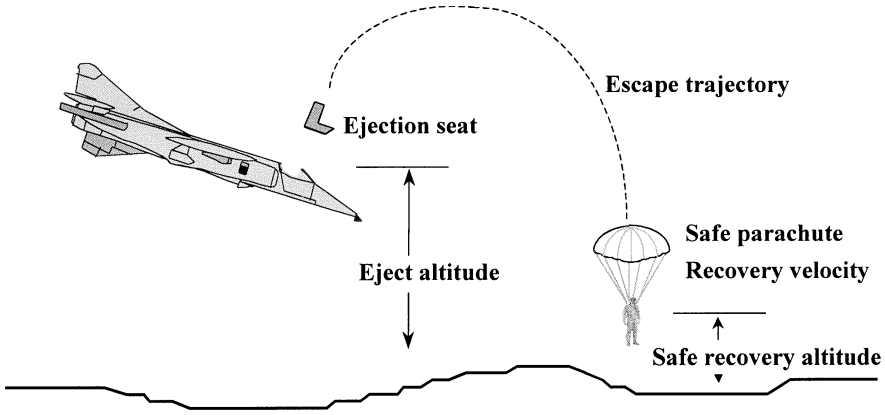


Fig. 2. Safe escape scenario.

input value results in a decrease in the output value and vice versa. In the ejection safety model problem in Section III, a positive causality index implies that the increase of the input parameter makes the ejection safer.

3) While a large causality index value indicates that the corresponding input has a greater impact on the output, a small one does not necessarily mean irrelevancy of the input parameter to the output. In addition, an input variable may have a small causality index if the relation between the output and the input is symmetric, (e.g., the roll angle at zero angular rates in our application in Section III). In this case, the causality index is positive in one half space, and negative in the other half space. The net average is zero, (or very close to zero since, practically, networks are never able to exactly model the true relation).

2) Minimization of False-Negative Error: Another important practical aspect in classification is to distinguish types of misclassification alarm. Not all alarms are treated equal in real world. For instance, let's consider the safe and unsafe classification cases. The first type of alarm, referred to as nuisance alarm, is encountered when a trained network declares an input unsafe although the correct classification should be safe. The second type, referred to as false negative, occurs if the trained network declares the incoming input vector the other way around. Clearly, the false negative alarm is of greater concern. Minimizing false negatives thus has a greater importance, even at the expense of a reasonable increase in nuisance alarm rate and added computational costs. The approach to achieving this is to introduce a penalty factor [31] that can skew the cost function emphasizing the danger of false negatives by using the weighting factor α

$$E = \frac{1}{2}\sum_{k=1}^{N_L} \alpha(t_k - y_k)^2 \tag{15}$$

where

$$\alpha = \begin{cases} 1 & \text{if } y_k > 0, t_k > 0 \text{ or } y_k < 0, t_k < 0 \\ F & \text{if } y_k < 0, t_k > 0 \\ M & \text{if } y_k > 0, t_k < 0 \end{cases} \tag{16}$$

TABLE I
FLIGHT PARAMETERS DETERMINING THE ESCAPE SAFETY

Parameter	Lower Limit	Upper Limit	Measurement resolution	Number of measurements
Pitch angle	-90 °	+90 °	30 °	6
Roll angle	-180 °	180 °	30 °	12
Flight Path Angle	-90 °	90 °	20 °	9
p	-180 °/sec	180 °/sec	30 °/sec	12
q	-180 °/sec	180 °/sec	30 °/sec	12
r	-180 °/sec	180 °/sec	30 °/sec	12
Altitude	0	1500 ft	50 ft	30
Velocity	0	450 keas	50 keas	9

The penalty factor is equal to 1 if classification is exact, but is equal to the constant M for false negatives, and is equal to F for nuisance alarms. To restrict false positives, M is greater than F. This technique is used in Section III-B-4.

Another way to bias the network decision toward more conservative safe decisions is to present more unsafe examples close to the decision boundary in the training set. Applying this idea on the query data has even a greater effect since the query data generally lies closer to the decision boundary. This is demonstrated by example in Section III-B-4, but is generally applicable.

III. AEROSPACE APPLICATION I: SAFE ESCAPE SYSTEM

In this section we present the application of QBL in determining the safe ejection envelope for a fighter jet [32], as illustrated in Fig. 2. We are trying to model the safe escape envelope as a function of airplane velocity, attitude, and other parameters known as initial conditions of an escape, as shown in Table I. It is a deterministic two-class decision problem. The ideal system should inform the pilot of the ejection safety status beforehand. Ejection safety can be simulated off-line via high fidelity ejection seat simulation, using EASY5 software [33]. One ejection simulation takes approximately 4 s on either a Silicon Graphics or a Pentium Pro 200 processor, which makes it invalid for on-flight real-time application. The main justification of using neural networks to replace the simulation software is the speed advantage needed in such life and death situation. Training data for NN simulations are supplied offline

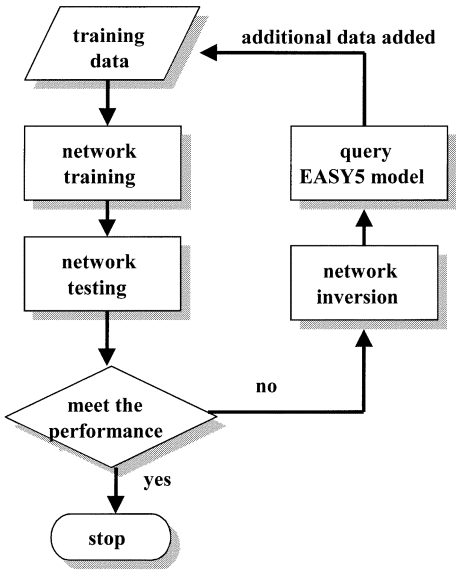


Fig. 3. QBL is used in an iterative manner to improve classification performance.

using EASY5 simulations. The goal is to predict the ejection safety based on the flight parameters.

The safety criteria is as follows: when the total recovery velocity becomes 50 ft/s, if the recovery altitude is higher than or equal to 50 ft, the ejection is declared as safe. There are eight flight parameters that affect the ejection safety. These parameters are the airplane attitude: pitch and roll angle, the flight path angle (FPA), the angular rates p, q, and r, the ejection altitude, and airplane speed.

Due to the high dimensionality of the input space, sampling with a reasonable resolution as shown in Table I would give $302\,330\,880\ (= 6*12*9*12*12*12*30*9)$ points. Generating this amount of data would take approximately 35 years.[2] Training a neural network using an advanced training algorithm like the NDEKF [31], [34]–[36] takes about one hour per 10 000 points using a Pentium Pro 200 processor. Thus, training the whole data set approximately needs an additional 3.5 years. This simply means that it is impossible to uniformly sample the input space with a reasonable resolution as suggested in Table I. Since, practically, we can only generate a tiny fraction of the necessary number of data, QBL strategy is applied in an iterative manner as depicted in Fig. 3 in order to efficiently generate and use the data.

A. CI as Sensitivity Measure of Input Components

The CI (14) has been calculated to evaluate the sensitivity of each input parameters and use it as a guidance to further eliminate noncontributing parameters. The sample results shown in Table II are typical CI values obtained under various network architectures, training strategy, and data selection. The results coincide with expert's assessment and demonstrate the value of using CI for data preparation. Since human experts consider the FPA and the *altitude* critical parameters in determining the safe escape, we expect them to exhibit higher positive CI values. In the mean time, the *velocity* of the fighter should exhibit negative CI values because the higher the velocity, the worse the safety of the escapee becomes. For the rest of the input parameters, the CI values may exhibit inconsistency due to either their negligible impact on the safety decision and/or due to their nonmonotonic relationship with the safety outcomes (see Section II-C-1).

Based on the CI calculations, we picked the three inputs with the largest CI values for the purpose of visualization as shown in Fig. 4. The plot shows little overlapping between the two classes (safe class and unsafe class). This means that with reduced dimensionality, the classification problem becomes well defined in the sense that the classification decision boundary is well separated and smooth.

B. Computer Experiment Results

For a feedforward multilayer network, the input components are scaled in [-1,1]. The dynamic range of individual input parameters is given in Table I. The network has one bipolar output indicating the safety of the ejection. The NDEKF method has been used for the training. As typical in NN application, we used three different data sets for training, validation, and testing. The validation set is presented to the network after each epoch during the training. The testing set is used after finishing the whole training. The network architectures have been determined by trial and error. The best results have been obtained with a single hidden layer network with five hidden neurons. All neurons have a bipolar sigmoid activation function described in (6).

1) A Comparison Case: In this section, we make a comparison between QBL and standard backpropagation (BP) learning. We fixed the angular rates to zero, and thus reduced the input space to five dimensions (note that only half of the roll angle range needs to be covered due to the output symmetry with respect to the roll angle at zero angular rates). This reduced space could then be uniformly sampled with a reasonably high resolution without QBL. Two cases have been compared. In the first case (Table IV), a neural network has been trained on a sparse data set with QBL. In the second case (Table V), the network has been trained on a high-resolution data set with standard BP learning.

For the QBL case, we randomly generated 9,900 data points and created three data sets (each contains 3300 points) for training, validation, and testing. We ran five computer simulations each with 9900 points generated using different random number generators. For the standard BP learning, we used 73 712 random data points for training. For each of the validation and testing sets in this case, we generated about 86 000 points (the actual number is slightly different due to convergence failure of the EASY5 steady-state analysis for some extreme initial ejection conditions) by uniformly sampling the input space using the resolution of Table III.

Another on-going research issue in QBL learning is when to start to introduce query data into the training set, i.e., when to stop the training and start network inversion. In this experiment, we used the following heuristics while evaluating the validation error.

[2]These processors are outdated. However, even using contemporary processors (in 2003) commonly available in the labs, data generation of 300 000 000 points using EASY5 is impractical.

TABLE II
THE NORMALIZED CI VALUES ARE CONSISTENTLY HIGH FOR THE FLIGHT PATH ANGLE, ALTITUDE AND VELOCITY

Case no.	No. hidden nodes	No. training epochs	No. training patterns	Sampling methods	Causality Index of Each of the 8 Inputs							
					pitch	roll	FPA	p	q	r	altitude	velocity
1	8	30	100	random	0.486	-0.58	2.338	-0.153	-0.696	-0.454	1.056	-0.366
2	5	415	461	QBL	0.099	0.239	2.501	-0.074	-0.058	-0.143	1.148	-0.575
3	5	40	100	random	-0.087	0.033	2.373	0.109	-0.044	0.288	1.385	-0.585
4	5	415	1000	random	0.016	0.219	2.12	-0.008	-0.021	0.031	1.73	-0.68

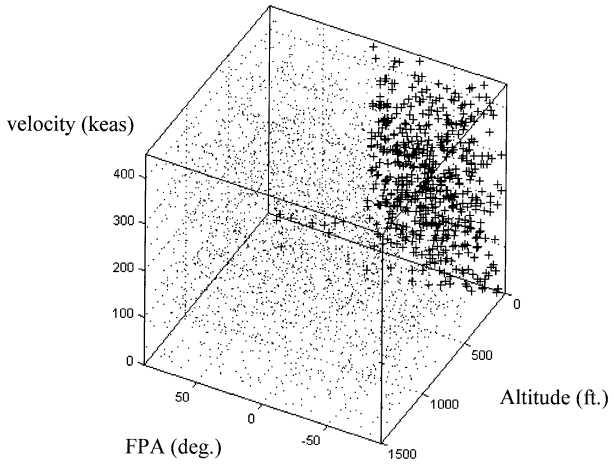


Fig. 4. Input data projection in 3-D. The classes are highly separable. Dots represent the safe class, while +s indicate the unsafe class.

TABLE III
HIGH RESOLUTION SAMPLING WITH ZERO ANGULAR RATES

Parameter	Lower Limit	Upper Limit	Measurement resolution	Number of samples
Pitch angle	-90 °	+90 °	30 °	6
Roll angle	0 °	180 °	30 °	6
Flight Path Angle	-90 °	90 °	20 °	9
Altitude	0	1500 ft	50 ft	30
Velocity	0	450 keas	50 keas	9

1) We observed that in order to benefit from QBL, the network needs to be partially trained while it forms a crude classification boundary, i.e., we should stop training prematurely. Therefore, we empirically chose to stop the training before 100 epochs (in a training scenario of 415 epochs total).
2) Using the epoch guideline set from heuristic 1 above, we trained the network until the validation error (rms sense) reached its local minimum and the error starts to increase. The validation error may have more than one local minimum within few epochs. In this case, we stop the learning after it reaches the last local minimum (see [37]).

Based on these heuristics, as shown in Fig. 5 and Fig. 6, the inversion took place near the last local minimum of the validation rms error at or before 100 epochs. At that point, using the network inversion target value of $\hat{t} = 0.0$, corresponding to the

TABLE IV
TESTING RESULTS OF THE 5-INPUTS COMPARISON CASE

Case Number	Test Error Count		
	No QBL	With QBL	Improvement
1	52	47	10%
2	49	43	12%
3	43	38	12%
4	61	42	31%
5	70	48	31%

TABLE V
TRAINING WITH A HIGH RESOLUTION DATA SET WITHOUT QBL

Case Number	Test Error Count		
	High Resolution	Low Resolution Average	Difference
1	1482	1579	6%

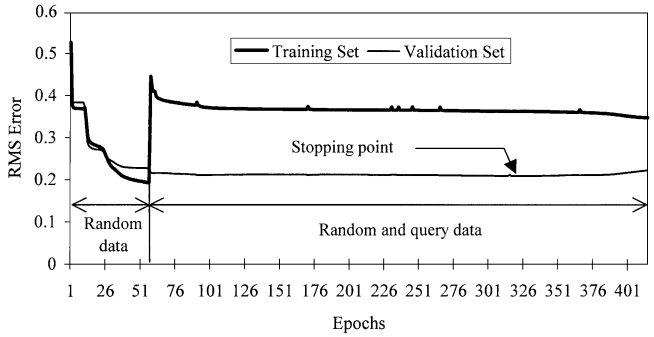


Fig. 5. Typical rms error performance for QBL learning. Network inversion was performed after 57 epochs. The training continued using the combined data set (original random data and queried data) for 261 more epochs where the testing is conducted.

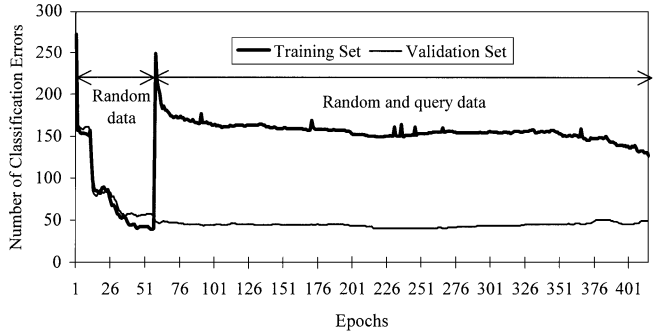


Fig. 6. Typical classification error count on both training and validation sets for QBL.

TABLE VI
RESULTS OF A TWO-STAGE QUERY-BASED LEARNING SCENARIO

Case number	Size of Training Set			Number of Training Epochs				Error Rate
	Uniform sampling	Inversion data	Total	With uniform data	After first query	After second query	Total	
1	100	0	100	40	0	0	40	11.75%
2	0	485	485	40	415	0	455	3.89%
3	0	461	461	40	415	415	870	3.32%

network decision boundary, we introduced inversion data into the training.

After adding the QBL data and continuing the network training, we also used heuristic 2 above for stopping (Heuristic 1 was not used since we do not want a partially trained network this time).

There are many ways of how to combine the query data with the original data (e.g., use only query data to continue training or add it to the original data) and whether to reset the network weights or not. We got the best result when the query data was added to the original random data. Then training was continued from the last state of weights.

Table IV and Table V summarize the test results. Using QBL improved the neural network performance by 19% in average. Table V compares the test results of the standard BP learning with 73 712 data points against the average of the five 3300 data points cases when trained without QBL (This time the 3300 data points cases are tested on the same high-resolution data set for the sake of a consistent comparison). With a much larger number of training data the improvement in testing data was only about 6%. From this experiment, we can see that QBL produced a much higher error improvement even though the number of data points used in QBL is significantly lower.

2) The Full Dimension Problem: Once we validated the use of QBL in the comparison case, we now deal with the original problem, where all the 8 input parameters are used. Unlike the 5-input case, where angular rates were ignored, we could not assume output symmetry with respect to the roll angle.

Since we were convinced that the QBL was found to be most effective when we start with a small and sparse data set in the previous section, we started the training with 100 randomly distributed training points. This is an extremely small amount compared to the total number of data needed to reasonably cover the entire search space (see Table I). We stopped training after 40 epochs where the validation error reached its minimum. Query data has then been generated, and training continued from the last state of weights. The network has then been inverted a second time, and the network underwent a second stage of QBL. At each stage the network was tested on 1234 test data and the error rate was calculated. The test data was generated in a fashion similar to the training data using an oracle, except that the test points were evenly distributed in the input space, while the training points were randomly distributed. Table VI shows that QBL lowered the test error by 67% in the first stage and after the second stage the overall error reduction was 72%.

In order to demonstrate the benefit of QBL statistically, here again we ran five simulations each with different training, validation, and test data. A training, validation, and testing set of 3000 uniform random samples each have been generated using the dynamic ranges given in Table VII. We used the previously discussed validation technique.

TABLE VII
SAMPLING RATES FOR THE FULL DIMENSION PROBLEM

Parameter	Lower Limit	Upper Limit	Measurement resolution	Number of samples
Pitch angle	-90 °	+90 °	90 °	2
Roll angle	-180 °	180 °	180 °	2
Flight Path Angle	-90 °	90 °	60 °	3
P	-180 °	180 °	120 °	3
Q	-180 °	180 °	120 °	3
R	-180 °	180 °	120 °	3
Altitude	0	1500 ft	500 ft	3
Velocity	0	450 keas	150 keas	3

TABLE IX
JITTER RESULTS ON MULTIPLE RUNS

Case Number	Test Error Count		
	QBL without Jitter	QBL with Jitter	Improvement
1	47	45	4%
2	43	41	5%
3	38	40	-5%
4	42	43	-2%
5	48	49	-2%
6	58	58	0%
7	73	75	-3%
8	83	85	-2%
9	67	67	0%
10	60	57	5%

TABLE VIII
TESTING RESULTS OF THE FULL DIMENSION (8 INPUT PARAMETERS) PROBLEM

Case Number	Test Error Count		
	No QBL	With QBL	Improvement
1	73	58	21%
2	81	73	10%
3	93	83	11%
4	77	67	13%
5	72	60	17%

Table VIII summarizes the test results of the full dimension problem with and without query. Using QBL improved the test by 14% in average.

3) Impact of Jitter on QBL Learning: Here we show the experimental results using jitter added to the inversion data as explained in Section II-A. We summarize the results in Table IX.

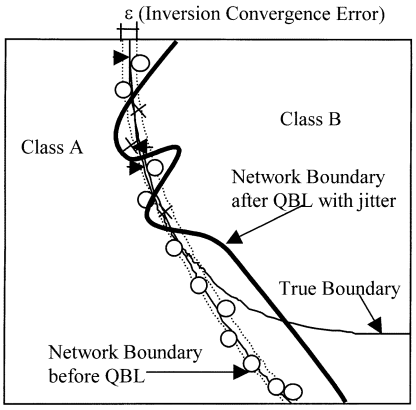


Fig. 7. Jitter can adversly affect training if we have only sparse data and the problem is well separable. The network overfits the decision boundary.

In the first five cases (case 1–case 5) the network had five inputs; the angular rates were removed. The last five cases used all input parameters—eight inputs. Random noise used is Gaussian with 0.0 mean and 0.001 standard deviation. The results of the ten runs with the infusion of jitter to the inversion data showed no improvement. This might be an indication that the decision boundary is relatively smooth and the classification boundary is separable (see Fig. 4). In addition, during the QBL process, we started with sparse training data and even after adding the query data the data set is still sparse. We believe that along a smooth and separated decision boundary, adding jitters to the inversion data may adversely impact the learning as shown in Fig. 7. This is due to the fact that the jitter unnecessarily forms a complex decision boundary (zigzag form, for instance) and forces the network to overfit, especially when the data is sparse.

4) Minimizing False Negatives: We use the penalty factor technique described in (15) and (16): $F = 1$, while M is varied as shown in Table X. The first two cases shown in the table are for a roughly trained network using 100 training samples. Introduction of the penalty factor decreased the number of false negatives and also decreased the total error rate. The results are consistent in the subsequent cases where larger training sets were used, up to the point where the total errors begin to increase in the last case. In all cases, we used again a validation set, this time stopping at the minimum of the false-negative errors rather than the total errors.

In Table XI we demonstrate the idea of minimizing false negatives by skewing the training data as described in Section II-C-2. In case 2 of the table, we generated 595 inversion data points, and added only the unsafe instances (234 points) to the original data set of case 1. We noticed a considerable decrease in the number of false negative alarms, at the expense of some increase in the total number of errors. The test set used below had 2048 samples.

IV. Aerospace Application II: Control Distribution

Difficult mapping problems occur routinely in engineering applications due to underspecification or control power limitations of physical systems. NNs have been shown to provide a compact and efficient means for implementing this type of mapping function for real-time simulation or embedded controller applications [38]. In the control loop of the ejection seat, it is desired to map desired moment components to achievable thrust commands to be applied to the four rocket nozzles. Control thrusts typically contain constraints that create a limited envelope of feasible moment combinations. This results in a mapping function that is not one to one, and therefore not directly invertible. The functional relationship of the moment and the thrust spaces is illustrated in Fig. 8.

The input for this problem is the set of desired moments to be applied to the body, shown as the range M of a noninjective function f. The output is the set of physically achievable thrusts satisfying the system constraints, represented by the inverse image of M under f, shown as T. A restriction of f to a subset of M, labeled L, is shown as the injective function h. The mapping of interest is $T = g(M)$. This is obtained by mapping M to L, and then using $T = h^{-1}(L)$. The mapping of points in M to L requires using pseudoinverse, iterative linear search, or other optimization procedures.These methods produce discrete data that describe the inverse mapping function g. Fig. 9 provides an illustration and block diagram description of the problem.

A 3-D moment envelope determined for a system with four rockets is shown in Fig. 10. The size and shape of this feasible envelope is defined by the individual and total thrust constraints given in (17) and (18). This is the set of moments depicted as the set L in Fig. 8.

$$\sum_{i=1}^{n} T_i = 6000 \tag{17}$$

and

$$0 \leq T_i \leq 3000 \tag{18}$$

where n is the number of nozzles and the thrust unit is lb. In addition, one of the nozzles may fail. In this case

$$T_j = 1500 \tag{19}$$

where j is the index of the failing nozzle. A linear optimization technique was used to generate desired moments to thrust data.

A. Results

A three-input-four-output multilayer feedforward network was used to map the moment components (Mx, My, Mz) into four thrust commands which can produce these moments, or the closest possible moments if the original ones are not feasible, according to the original data generated by the oracle. Two hidden layers were used containing 10 neurons in the first hidden layer and five in the second layer. QBL was used to emphasize relearning of data sets with large errors. The process goal is to keep the absolute testing error within the required upper limit for all test patterns.

The multilayer feedforward network was trained using 1331 initial learning data. The data were generated by dividing the input space into 10 divisions along each dimension, then adding some noise to the step size. The training set had high noise level,

TABLE X
TESTING RESULTS USING A PENALTY FACTOR

Case Number	M	Size of Training Set	False Negatives	Nuisance Alarms	Total Errors
1	1	100	6.81%	7.2%	14.01%
2	1.5	100	4.86%	7.62%	12.48%
3	1	2012	1.12%	1.12%	2.24%
4	1.5	2012	0.93%	1.27%	2.2%
5	2.0	2012	1.32%	1.66%	2.98%

TABLE XI
REDUCING FALSE NEGATIVES BY USING ONLY UNSAFE SAMPLES IN THE QUERY DATA

Case Number	Size of Training Set			Number of False Negatives	Number of Nuisance Alarms	Total Number of Errors
	Uniform Sampling	Inversion Data	Total			
1	2012	0	2012	23	23	46
2	2012	234	2246	15	47	62

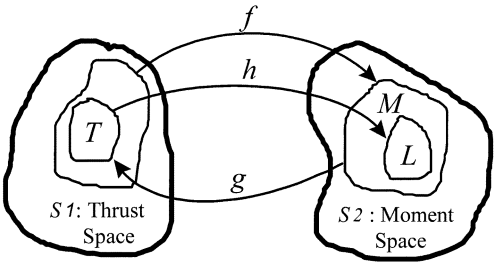


Fig. 8. Mapping between thrust and moment spaces. M is the desired set of moments. L is the achievable set of moments by the corresponding constrained set of thrusts T.

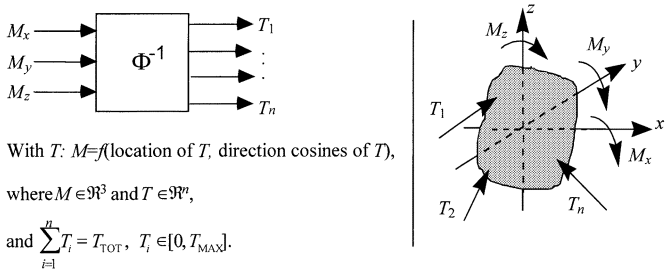


Fig. 9. Nonlinear moments to thrusts mapping. The inverse transform Φ^{-1} is nonlinear due to the total thrust and individual maximum thrust constraints.

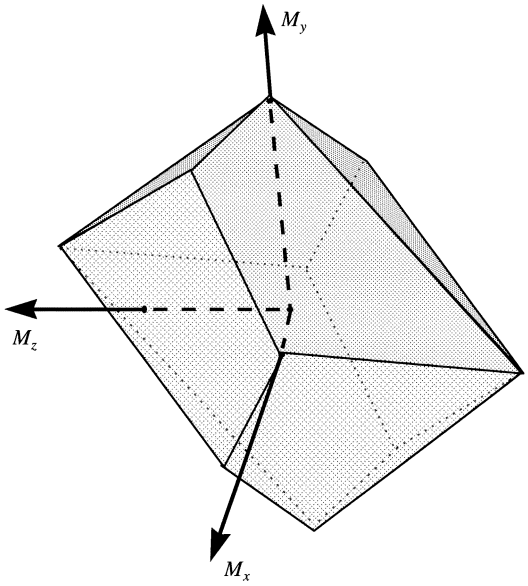


Fig. 10. Achievable moment envelope for the control of a four-rocket ejection seat, with all four nozzles working.

the validation set (used during training) had medium noise level, and the test set had no noise. Thus, the three data sets have similar distribution. The NDEKF algorithm was used for training the network.

Test results revealed that though the error of the first and third outputs (T_1 and T_3) were maintained within the same range, the errors of T_2 and T_4 exhibited several spikes where the error exceeded 500 lb, as shown in Fig. 11(a). All these spikes happened during the first 500 points of the test set. T_4 had 26 points with error larger than 500 lb, while T_2 had 24 points, always occurring simultaneously with corresponding T_4 points. Using the 26 points where T_4 error exceeded 500 lb, as target values, query data was generated a first time, using network inversion as described above, resulting in 127 new training data. Using the oracle, corresponding real outputs have been found. The newly generated 127 patterns were uniformly distributed in the original training data set. 127 original patterns were removed so that the length of the data sets would stay fixed to 1331. This would allow comparison of results based on the quality of data, and eliminating the effect of data set length. The network was then retrained, restarting with initial random weights. After this first query cycle, re-testing the network showed a reduction in the number of points of larger error to only 19 for both T_2 and T_4. A second inversion and query have been done using these 19 points resulting in 96 new training data points. These points were added to the 127 points obtained from the first query cycle and the total of 223 data points have been incorporated with the original data in the same fashion as in the first cycle. Using this third data set, the network has been retrained again restarting with initial random weights. Test results showed further reduction in the number of error spikes to 18 and 17 points in T_2 and T_4, respectively. Fig. 12 shows the decrease in number of error spikes after the first and second application of QBL.

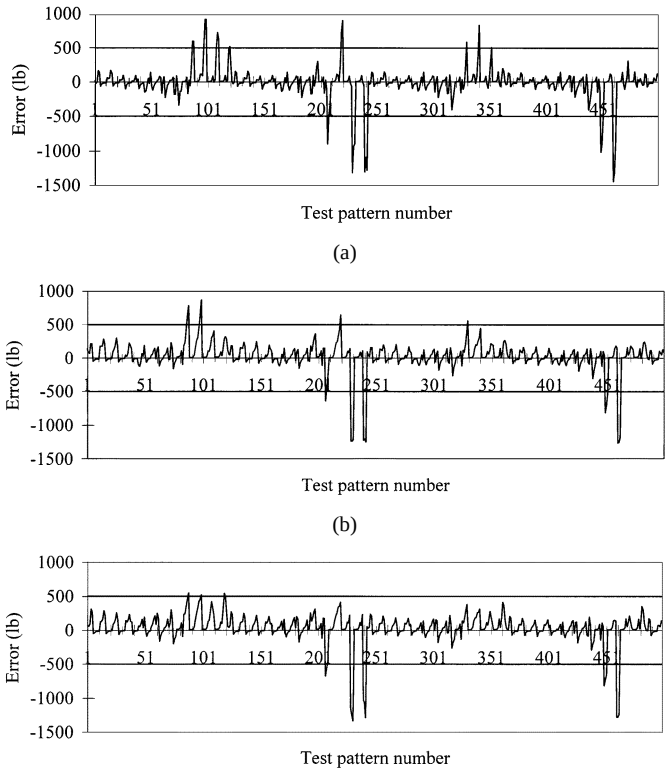


Fig. 11. (a) Error of T_4 before using QBL. (b) Error of T_4 after first use of QBL. (c) Error of T_4 after second use of QBL. The number of error spikes is decreased.

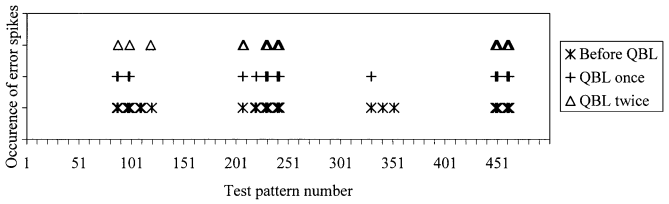


Fig. 12. Occurrence of error exceeding 500 lb. The number of occurrences is decreased from 26 to 19 after first application of QBL and again to 17 after second application of QBL.

Fig. 11 shows the progress made in testing error. After the first query, the testing error is significantly reduced. After the second query, most of the positive error spikes have been eliminated, as shown in Fig. 11(c). Investigation of applying the QBL technique to target reduction of the negative error values is on-going.

The backpropagation algorithm has also been applied to train the feed-forward network (with 10-5 hidden units). Overall results seem to agree with that of the NDEKF algorithm. Fig. 13 shows typical learning curves for training data and validation data sets. Various different learning gains, momentum gains, and random seed values were tried. The convergence speed for the QBL method is drastically improved. The maximum error in the figures is the maximum error among 4 outputs.

V. Conclusion

QBL facilitates maximum utilization of training data, which is necessary for problems where data generation is expensive.

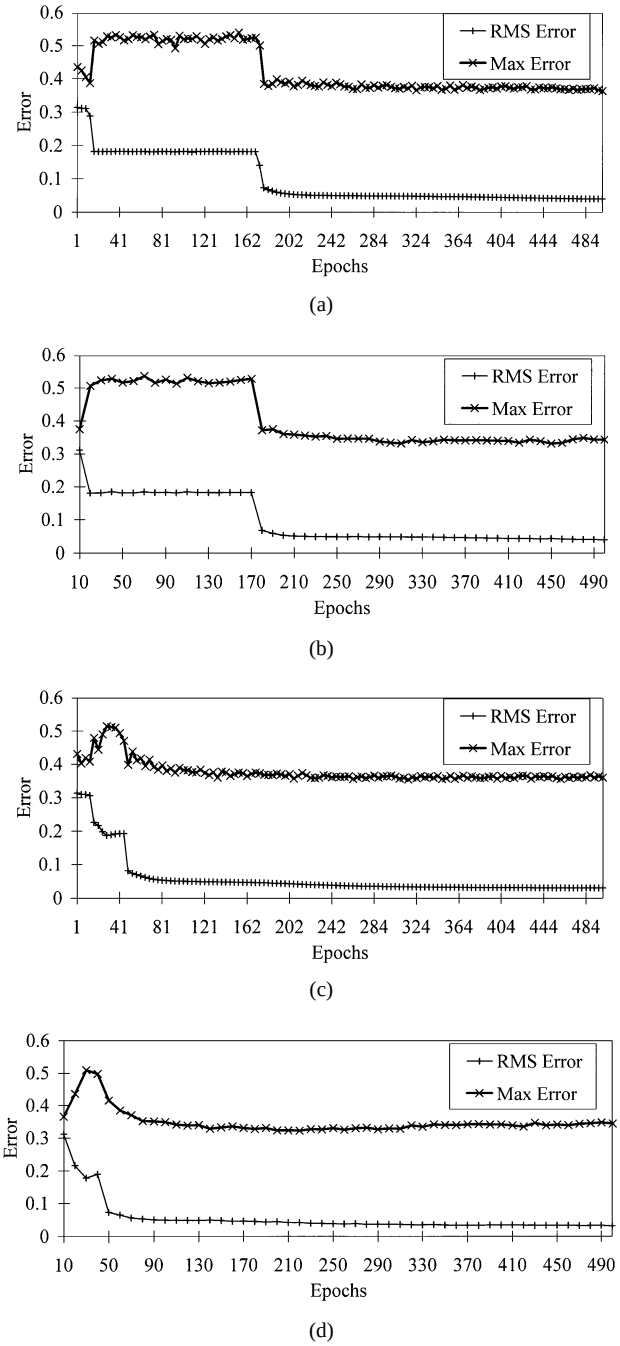


Fig. 13. Backpropagation RMS and maximum errors. (a) Learning curve of the training data set without query set. (b) Testing error conducted with the validation set for every 10 epochs of training without query set. (c) Learning curve of the training data set with query data. (d) Testing error conducted with the validation set for every 10 epochs of training with query data.

We have shown the effectiveness of QBL in two aerospace applications. QBL was successfully applied to a classification problem as well as to a continuous mapping problem. When to invert the network and how to incorporate newly generated query data into training processes are on-going research topics. In the classification problem, the neural network was inverted to find inputs along the network decision boundary. We validated the use of QBL using a comparison case. QBL has improved the network performance by 19% in the comparison case and by 14% in the full-dimension problem. We also showed

that after querying the oracle, data obtained from only one class could be used to minimize false negatives. In continuous mapping problems, choosing the inversion target value is more complicated. In our control distribution problem, the goal was to reduce the maximum error. This is also a primary objective in many other engineering problems. Systems can have envelopes of operation that make certain input patterns occur more frequently than others. If errors exist at these frequent input patterns, performance may drastically degrade. The inversion target values can be chosen for input data points that exhibit such degradation. We used the corresponding network output values of those input data points as the inversion targets. We also have shown the possibility to reduce the maximum positive error using QBL. In utilizing newly acquired query data, one can use both the original and the query data and re-train the network. One can also retrain the network with the query data only. Additionally, the network can be incrementally retrained (i.e., retraining the network from previous weights), or can be reinitialized before retraining.

In addition, we have analyzed the use of the CI to evaluate the sensitivity of the input variables and choosing the most influential ones. We have also experimentally shown the effect of adding jitter to the query data. Although, the evidence may not be extensive, we believe that adding jitters may adversely impact the training for the classification task when having a well-separable and smooth boundary, especially, when only a small number of data is available along the boundary. Added jitters may create more complicated decision landscape and force the network to overfit.

In combination, the techniques herein can allow significantly enhanced performance, for these examples and a variety of similar applications.

REFERENCES

[1] J. Hwang, J. Choi, S. Oh, and R. Marks, II, "Query-based learning applied to partially trained multilayer perceptrons," *IEEE Trans. Neural Networks*, vol. 2, pp. 131–136, Jan. 1991.

[2] ——, "Query learning based on boundary search and gradient computation of trained multilayer perceptrons," in *Proc. Int. Joint Conf. Neural Networks*, San Diego, 1990, pp. 57–62.

[3] S. Oh, R. Marks, II, and M. El-Sharkawi, "Query based learning in a multilayered perceptron in the presence of data jitter," in *Proc. First Int. Forum on Applications of Neural Networks to Power Systems*, pp. 57–62.

[4] J. Hwang and H. Li, "Interactive query learning for isolated speech recognition," in *Proc. IEEE-SP Workshop on Neural Networks for Signal Processing II*, pp. 93–102.

[5] D. Cohn, "Neural network exploration using optimal experiment design," *Neural Networks*, vol. 9, no. 6, pp. 1071–1083, 1996.

[6] K. Fukumizu, "Statistical active learning in multilayer perceptrons," *IEEE Trans. Neural Networks*, vol. 11, 2000.

[7] V. Fedorov, *Theory of Optimal Experiments*. New York: Academic, 1972.

[8] D. MacKay, "Information-based objective functions for active data selection," *Neural Computation*, vol. 4, no. 4, pp. 589–603.

[9] J. Ratsaby, "Incremental learning with sample queries," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 20, pp. 883–888, 1998.

[10] C. Christian, "Pedagogical pattern selection strategies," *Neural Networks*, vol. 7, no. 1, pp. 175–181, 1994.

[11] D. Angluin, "A note on the number of queries needed to identify regular languages," *Inform. Contr.*, vol. 51, pp. 76–87, 1982.

[12] D. Cohn, L. Atlas, and R. Ladner, "Training connectionist networks with queries and selective sampling," in *Advances in Neural Information Processing Systems 2*, D. Touretzky, Ed. Cambridge, MA: MIT Press, 1990.

[13] E. Baum and K. Lang, "Neural network algorithms that learn in polynomial time from examples and queries," *IEEE Trans. Neural Networks*, vol. 2.

[14] A. Linden and J. Kindermann, "Inversion of multilayer nets," in *Proc. Int. Joint Conf. Neural Networks*, vol. II, Wash., DC, 1989, pp. 425–430.

[15] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Englewood Cliffs, NJ: Prentice-Hall, 1998.

[16] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proc. IEEE*, vol. 7, pp. 1550–1560, 1990.

[17] R. Reed, R. J. Marks, II, and S. Oh, "Similarities of error regularization, sigmoid gain scaling, target smoothing, and training with jitter," *IEEE Trans. Neural Networks*, vol. 6, pp. 529–538, 1995.

[18] R. Reed, S. Oh, and R. J. Marks, II, "An equivalence between sigmoidal gain scaling and training with noisy (jittered) input data," in *Proc. RNNS/IEEE Symp. Neuroinformatics Neurocomputing*, Rostov-on-Don, Russia, 1992.

[19] D. C. Plaut, S. J. Nowlan, and G. E. Hinton, "Experiments on learning in a multilayered perceptron in the presence of data jitter," Carnegie-Mellon Univ., Tech. Rep. CMU-CS-86-126, 1986.

[20] J. L. Elman and D. Zipser, "Learning the hidden structure of speech," *J. Accoust. Soc. Amer.*, vol. 83, no. 4, pp. 1615–1626, 1988.

[21] P. Refregier and J. M. Vignolle, "An improved version of the pseudo-inverse solution for classification and neural networks," *Europhys. Lett.*, vol. 10, no. 4, pp. 387–392, 1989.

[22] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, "Generalization by weight-elimination applied to currency exchange rate prediction," in *Proc. Int. Joint Conf. Neural Networks*, vol. I, Seattle, WA, 1991, pp. 837–841.

[23] J. Sietsma and R. J. F. Dow, "Creating artificial neural networks that generalize," *Neural Networks*, vol. 4, no. 1, pp. 67–79, 1991.

[24] S. Oh, R. J. Marks, II, and M. A. El-Sharkawi, "Query based learning in a multilayered perceptron in the presence of data jitter," in *Applications of Neural Networks to Power Systems*, M. S. El-Sharkawi and R. J. Marks, II, Eds. New York: IEEE, 1991, pp. 72–75.

[25] C. H. Séquin and R. D. Clay, "Fault tolerance in feed-forward artificial neural networks," in *Neural Networks: Concepts, Applications, and Implementations*, P. Antognetti and V. Milutinović, Eds. Englewood Cliffs, NJ: Prentice-Hall, 1991, vol. IV, pp. 111–141.

[26] J. I. Minnix, "Fault tolerance of the backpropagation neural network trained on noisy inputs," in *Proc. Int. Joint Conf. Neural Networks*, vol. I, Baltimore, MD, 1992, pp. 847–852.

[27] D. A. Cohn, Z. Ghahramani, and M. I. Jordan, "Active learning with statistical models," *J. Artific. Intell. Res.*, vol. 4, pp. 129–145, 1996.

[28] K. Baba, I. Enbutu, and M. Yoda, "Explicit representation of knowledge acquired from plant historical data using neural network," in *Proc. Int. Joint Conf. on Neural Networks*, vol. II, Washington, 1992, pp. 579–583.

[29] Z. Boger, "Knowledge extraction from artificial neural networks models," in *Proc. IEEE Int. Conf. on Syst., Man and Cybern.*, vol. 4, Orlando, FL, 1997, pp. 3030–3035.

[30] B. Mak and R. Blanning, "An empirical measure of element contribution in neural networks," *IEEE Trans. Syst., Man and Cybern.*, vol. 28, pp. 561–564, 1998.

[31] E. W. Saad, D. V. Prokhorov, and D. C. Wunsch, II, "Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks," *IEEE Trans. Neural Networks*, vol. 9, no. 6, 1998.

[32] E. W. Saad, J. J. Choi, J. L. Vian, and D. C. Wunsch, II, "Efficient training techniques for classification with vast input space," in *Proc. Int. Joint Conf. Neural Networks*, Wash., DC, 1999.

[33] *EASY5 User's Guide*, The Boeing Company, Seattle, WA, 1997.

[34] S. Singhal and L. Wu, "Training multilayer perceptrons with the extended Kalman algorithm," in *Advances in Neural Information Processing Systems 1*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 133–140.

[35] G. V. Puskorius and L. A. Feldkamp, "Roles of learning rates, artificial process noise and square root filtering for extended Kalman filter training," in *Proc. Int. Joint Conf. Neural Networks*, Wash., DC, July 1999.

[36] ——, "Decoupled extended Kalman filter training of feedforward layered networks," in *Proc. Int. Joint Conf. Neural Networks*, vol. I, Seattle, WA, 1991, pp. 771–777.

[37] L. Prechelt, "Automatic early stopping using cross validation: quantifying the criteria," *Neural Networks*, vol. 11, no. 4, pp. 761–767, 1998.

[38] J. L. Vian and N. A. Visnevski, "Neural Networks for Nonlinear Control Distribution," in *Proc. Int. Federation of Automatic Control: 3rd Symp. Intelligent Components and Instruments for Control Applications*, Annecy, France, 1997, pp. 229–234.

Emad W. Saad (S'95–M'00) received the B.S. degree with Honors from Ain Shams University, Cairo, Egypt, in 1992. He received the M.S. and Ph.D. degrees in electrical engineering in 1996 and 1999, respectively, from the Applied Computational Intelligence Laboratory, Texas Tech University, Lubbock.

Since October 2003, he has been a Postdoctoral Fellow with the Research Center for Advanced Manufacturing, Southern Methodist University, Richardson, TX, where he develops neural network and signal and image processing applications for sensing and control of manufacturing processes. Prior to that, he was with MCI, Richardson, TX, since 2000, where he developed network optimization algorithms using genetic algorithms, graph theory, and operations research. Prior to his position with MCI, he was with Deep View Systems, West Bloomfield, MI, in 1999, where he worked on neural networks and data mining for financial and business applications. In the summers of 1997 and 1998, he was with The Boeing Company, Seattle, WA, under a contract with Texas Tech University, where he was working on neural networks with query-based learning and computational applications for aerospace decision and control problems. His research interests include pattern recognition, time series prediction, signal processing, neurocontrol, adaptive critic designs, optimization, and data mining.

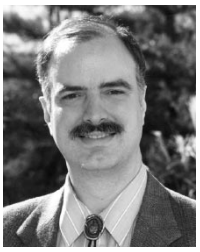
Jai J. Choi received the B.S. and M.S.E. degrees in electronics engineering from Inha University, Inchon, Korea, in 1979 and 1981 and the M.S.E.E. and Ph.D. degrees in electrical engineering from the University of Washington, Seattle, WA, in 1987 and 1990, respectively.

In 1990, he joined The Boeing Company, Seattle, where he is currently an Associate Technical Fellow with the Mathematics and Computing Technology Group, Computer Science Department, Phantom Works. His research is in the area of intelligent information processing and management. His expertise includes adaptive signal processing and control, neural networks and fuzzy logic, wavelet analysis, machine learning, computer security analysis, adaptive system modeling, and manufacturing intelligence. Since 1992, he has been an Affiliate Professor and graduate faculty member with the electrical engineering department, University of Washington, Seattle. He is also an adjunct faculty member at Henry Cogswell College, Everett, WA.

He is currently an Associate Editor for the IEEE TRANSACTIONS ON NEURAL NETWORKS. He has organized and chaired sessions in various international conferences including IEEE World Congress on Computational Intelligence, IEEE International Conference on Neural Networks, IEEE International Symposium on Circuits and Systems, and IEEE International Joint Conference on Neural Networks, and others.

John L. Vian (M'87–SM'91) received the B.S. degree in mechanical engineering from Purdue University, West Lafayette, IN, in 1981, and the M.S. degree in aeronautical engineering and the Ph.D. degree in electrical engineering from Wichita State University, Wichita, KS, in 1986 and 1991, respectively.

He is a Technical Fellow with Boeing Phantom Works, Seattle WA, where he is currently responsible for leading enabling technology research in diagnostics, prognostics, and vehicle health management. His experience is in flight control, safety-critical decision and control systems, dynamic modeling and simulation, and systems health. He has supported flight control design on KC-135, Boeing Robotic Air Vehicles, DARPA STOVL Aircraft, and Boeing 777 programs, and has served as Principal Investigator on multiple government research contracts. He is a licensed professional engineer, and has three patents and more than 25 technical publications. He serves as an ABET electrical engineering program evaluator, National Science Foundation panelist, and teaches at Henry Cogswell College in Everett WA.



Donald Wunsch, II (SM'94) completed a Humanities Honors Program at Seattle University in 1981. He received the B.S. degree in applied mathematics from the University of New Mexico in 1984, the M.S. degree in applied mathematics from the University of Washington in 1987, and the Ph.D. degree in electrical engineering in 1991, respectively.

Since July 1999, he is the Mary K. Finley Missouri Distinguished Professor of Computer Engineering in the Department of Electrical and Computer Engineering, University of Missouri-Rolla. He heads the Applied Computational Intelligence Laboratory and also has a joint appointment in Computer Science. Previously, he was Associate Professor of Electrical and Computer Engineering, and Computer Science, at Texas Tech University. Prior to joining Texas Tech in 1993, he was a Senior Principal Scientist with Boeing, where he invented the first optical implementation of the ART1 neural network, featured in the 1991 Boeing Annual Report, and other optical neural networks and applied research contributions. He has also worked for International Laser Systems and Rockwell International, and consulted for Sandia Labs, White Sands Missile Range, Texas Tech, Boston University, and Accurate Automation Corporation. Research activities include adaptive critic designs; neural network pattern analysis, optimization, forecasting and control; computer security; bioinformatics; financial engineering; fuzzy risk assessment for high-consequence surety; intelligent agents; graph theory; quantum logic; and Go. He is heavily involved in research collaborations with former Soviet scientists. He is an Academician in the International Academy of Technological Cybernetics, and in the International Informatization Academy.

Dr. Wunsch is the recipient of the Halliburton Award for excellence in teaching and research at Texas Tech, and a National Science Foundation CAREER Award. He is a member of the International Neural Network Society, Association for Computing Machinery, Society of Photo Instrumentation Engineering, Phi Kappa Phi, a life member of the American Association of Artificial Intelligence, a life member of Sigma Xi, and previously served as an Associate Editor of the IEEE TRANSACTIONS ON NEURAL NETWORKS and voting member of the IEEE Neural Network Council. He was elected to the INNS Board of Governors for 2002—present, and served as Technical Co-Chair for IJCNN 02 and General Chair for IJCNN 03.