INCREMENTAL LEARNING OF ULTRASONIC WELD INSPECTION SIGNALS

R. Polikar, L. Udpa, S.S. Udpa

Materials Assessment Research Group, Dept. Of Electrical and Computer Engineering
Iowa State University, Ames, IA 50011, USA

Abstract. In this paper, we present LEARN++, a new algorithm that allows existing classifiers to learn incrementally from new data without forgetting previously acquired knowledge. LEARN++ is based on generating multiple classifiers, each trained with a different subset of the training data and then combining them to form an ensemble of classifiers using weighted majority voting. The fundamental contribution of the algorithm lies in the manner in which the training data is partitioned. The results demonstrate the feasibility and effectiveness of the approach in learning from new data.

INTRODUCTION

Automated signal classification (ASC) schemes for characterization of nondestructive evaluation (NDE) signals are becoming increasingly popular due to large volumes of data that need to be analyzed in an accurate, efficient and systematic manner. Applications of such systems include identifying defects in aircraft engines and wheels [1], in tubings and pipings of nuclear power plants [2,3], in artificial heart valves, etc. Currently used ASC schemes, however, do not allow incremental learning of additional data particularly when the new data introduces additional classes. When new data become available, most existing ASC schemes are reinitialized and retrained using a combination of previously used data and the new data. This results in all previous learning to be lost, a phenomenon known as *catastrophic forgetting*.

Various real world scenarios in NDE applications warrant incremental learning algorithms. For example, in nuclear power plants, data are collected from various tubings or pipings during different outage periods, and new types of defect and geometry indications can be discovered in aging components. Classification algorithms developed using previously collected databases may then become inadequate in successfully identifying new types of indications.

In this paper, we present a new algorithm, called LEARN++, which allows common supervised neural network classifiers to learn incrementally from new data. The algorithm has been applied to a large number of databases, including synthetic benchmark datasets, as well as real world datasets. In this paper we present the classification performance of the algorithm on ultrasonic signals obtained from nuclear submarine hulls. Applications to other databases can be found in [4].

INCREMENTAL LEARNING

Ensemble of Classifiers

Incremental learning has recently been a topic of active research interest in machine learning and artificial intelligence, and therefore, several variations of this problem have been addressed in the literature [4]. For example, in one extreme case, incremental learning from new data is trivialized by allowing retraining using a combination of original training data and the new data, with no additional classes introduced. On the other extreme end, incremental learning algorithm is required to learn in an on-line setting, where the learning is carried out on an instance-by-instance basis with some instances introducing new classes. In this paper, we define an incremental learning algorithm as one that is capable of learning additional information from new data, which may include new classes, without forgetting prior knowledge and without requiring access to previously used data. LEARN++, proposed in this paper is an algorithm for supervised neural networks (NN), satisfying these criteria.

LEARN++ is inspired by Schapire's *adaptive boosting (AdaBoost)* algorithm [5], which was originally proposed for improving the accuracy of weak learning algorithms. Boosting is based on a majority voting of hypotheses (classification rules) which are generated by a weak learner using different distributions of the training data. Littlestone *et al.* have shown that the *weighted majority algorithm*, which assigns weights to different hypotheses based on an error criterion to construct a compound hypothesis, performs better than any of the individual hypotheses [6]. They also showed that the error of the compound hypothesis is closely linked to the error bound of the best hypothesis. In essence, both AdaBoost and LEARN++ employ a weighted combination of a group of classifiers, rather than using just one classifier as conventional classification algorithms do. The weights of each classifier are determined based on the individual performance of the classifier on its own training dataset.

The idea of generating an ensemble of classifiers is not new, as it has been explored by a number of researchers. Wolpert first introduced the idea of combining hierarchical levels of classifiers, using a procedure called *stacked generalization* [7]. Kitler *et al.* analyzed error sensitivities of various voting and combination mechanisms [8], and Rangarajan *et al.* investigated the capacity of voting systems [9]. Ji and Ma proposed an alternative approach of combining classifiers by generating simple perceptrons of random parameters and then combining the perceptron outputs using majority voting [10]. The motivation behind this approach was to obtain time and space efficiency, as well as good performance, since AdaBoost is computationally quite expensive. An excellent overview of ensemble of classifiers can be found in [11], whereas a review of comparing ensemble of classifiers with other types of learners can be found in [12].

The steady increase in research efforts on combining classifiers has been mostly limited to improving the performance of classifiers, leaving the viability of using such an approach for incremental learning unexplored. LEARN++ has emerged as a result of investigating the feasibility of ensemble of classifiers for incremental learning.

Learn++: An Incremental Learning Algorithm for Neural Networks

Figure 1 illustrates the algorithm LEARN++, which runs iteratively for each new dataset $\mathcal{D}_k$ that becomes available to the classifier. The inputs to LEARN++ include a sequence of m training instances, $S=[(\mathbf{x}_1,y_1),(\mathbf{x}_2, y_2),\ldots,(\mathbf{x}_m,y_m)]$, a weak learning algorithm, **WeakLearn**, and an integer T_k specifying the number of iterations.

```
Algorithm LEARN++
Input: For each dataset $\mathcal{D}_k$ $k=1,2,...,K$ that becomes available
    • Sequence of $m$ examples $S=[(x_1,y_1),(x_2,y_2),...,(x_m,y_m)]$.
    • Weak learning algorithm WeakLearn (MLP).
    • Integer $T_k$, specifying the number of iterations.
Do for $k=1,2,...,K$:
  Initialize $D_1(i)=1/m, \forall i$.
  Do for $t=1,2,...,T_k$:
    1. Randomly choose training $TR_t$ and testing $TE_t$ subsets from $D_t$.
    2. Call WeakLearn, providing it with $TR_t$ from distribution $D_t$.
    3. Get back hypothesis $h_t : X \rightarrow Y$, and calculate its error $\varepsilon_t = \sum_{i:h_t(x_i)\neq y_i} D_t(i)$ on
       $TR_t + TE_t$. If $\varepsilon_t > 1/2$, set $T = t-1$, discard $h_t$ and go to step 1. Otherwise,
       compute scaled error as $\beta_t=\varepsilon_t/(1-\varepsilon_t)$.
    4. Call weighted majority, obtain the overall hypothesis
       $H_t = \arg\max_{y\in Y} \sum_{t:h_t(x)=y} \log\frac{1}{\beta_t}$, and compute the overall error $E_t = \sum_{i:H_t(x_i)\neq y_i} D_t(i)$
    5. Set $B_t = E_t/(1-E_t)$, and update distribution $D_t$:
       $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} B_t, & if\ H_t(x_i)=y_i \\ 1, & otherwise \end{cases}$ where $Z_t = \sum_i D_t(i)$ is a
       normalization constant such that $D_{t+1}$ will be a distribution
Call weighted majority and Output the final hypothesis:
    $h_{final}(x) = \arg\max_{y\in Y} \sum_k \sum_{t:h_t(x)=y} \log\frac{1}{\beta_t}$
```

Figure 1. Algorithm LEARN++

For each dataset $\mathcal{D}_k$ that becomes available, LEARN++ starts by initializing a distribution function $D_1(i)=1/m$, $i=0,1,...,m$ on current dataset, such that the training instances which will be drawn according to this distribution will have equal likelihood to be selected into the first training set. LEARN++ then enters into an iterative loop, where at each iteration t, a new hypothesis is generated. During the t^{th} iteration, LEARN++ selects a training dataset TR_t and a test dataset TE_t according to the current distribution D_t in step 1. In step 2, the weak learning algorithm **WeakLearn** is trained using the training dataset TR_t. A classification rule, h_t, is obtained as the t^{th} hypothesis in step 3. Also in this step, ε_t, the error of h_t is obtained by evaluating the hypothesis on all instances in TR_t and TE_t. If this error is greater than half, current h_t is discarded, and the algorithm returns to step 1 to select a different training dataset. Otherwise, a normalized error term β_t is computed. Note that, since ε_t is between 0 and ½, β_t will be between 0 and 1. In the fourth step, LEARN++ calls the weighted majority algorithm to compute the composite hypothesis, H_t. The weighted majority computes the total vote each class receives, from all previous t hypotheses. Each vote is weighted by the inverse log of normalized error of that hypothesis. This ensures that hypotheses that did well on their own training data are weighted more heavily than those that did not do very well. The class that receives the highest total vote becomes the classification rule of the t^{th} composite hypothesis H_t. The error E_t and the normalized error B_t of the composite hypothesis are also computed. Finally, in step 5, LEARN++ updates the distribution D_t according to the performance of H_t. The distribution update rule decreases the weight of all instances that are correctly classified by H_t, such that they are less likely to be selected into the next training dataset. Therefore, as the algorithm proceeds, increasingly difficult examples are selected into the training dataset. This procedure allows rapid learning of new data, since only those instances of the new dataset that are misclassified by the classifier are used for further training the classifier.

After a pre-specified number of hypotheses are generated, LEARN++ computes the final hypothesis, using a weighted majority voting on all hypotheses generated up to that point. This algorithm was implemented and tested using a large database of ultrasonic weld inspection signals obtained from nuclear submarine hull weldings. The procedure for obtaining this dataset is briefly discussed next, followed by the results.

ULTRASONIC INSPECTION OF SUBMARINE HULL WELDINGS

Welding regions are often susceptible to various kinds of defects due to imperfections introduced into the material during the welding process. In submarine hulls, such defects manifest themselves in the forms of crack, porosity, slag, lack of fusion (LOF), and incomplete penetration. Among these, cracks and LOFs cause the most serious threat, since they can eventually cause structural damages if remedial actions are not taken.

Non-destructive techniques used commonly to inspect these welds are ultrasonic testing and radiographic testing. Each technique has its own advantages and disadvantages; however, ultrasonic testing is known to be more sensitive to planar defects, which are the more dangerous ones. Ultrasonic testing is conducted by using a piezoelectric transducer, which launches an ultrasonic wave of 1 ~ 5 MHz, into the material. Discontinuities along the path of the ultrasonic wave produce a reflected signal to be received by the transducer, which can then be recorded and analyzed. Figure 2 conceptually illustrates the ultrasonic testing procedure.

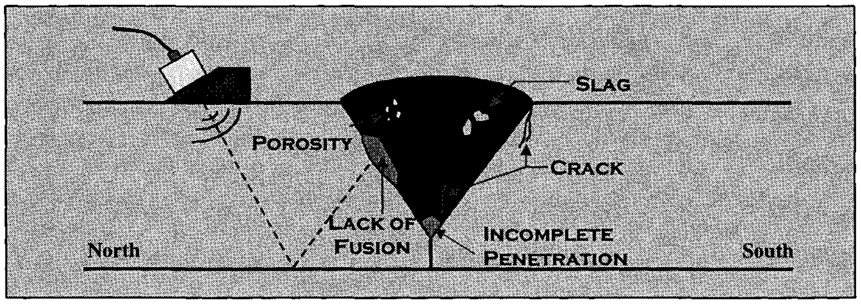

Figure 2. Ultrasonic testing of submarine hull welds

RESULTS AND DISCUSSIONS

The goal of the classification algorithm in this application is the identification of four different types of defects, namely, crack, porosity, slag and LOF, from the discrete wavelet transform coefficients of the ultrasonic A-scans. A total of 156 C-scans were obtained by examining 78 hull plates from both sides of the weld. 106 C-scans were randomly selected to be used for training, and 50 were selected to be used for validation. From the C-scans reserved for training, 2200 A-scans, each of 512-point length, were randomly selected for training and 800 were selected for testing (different than the validation database). 149-point DWT coefficients were computed for each A-scan to be used as feature vectors. The training instances were further divided into three subsets to simulate three different databases that become available at different times. Furthermore, additional classes were added to the later datasets, to test the incremental learning ability of the algorithm. Table 1 shows the distribution of the instances in various datasets, whereas Figure 3 illustrates typical signals from these four classes.

Table 1. Distribution of weld inspection signals

	CRACK	LOF	SLAG	POROSITY
SI	300	300	0	0
S2	150	300	150	0
S3	200	250	250	300
TEST	200	300	200	100

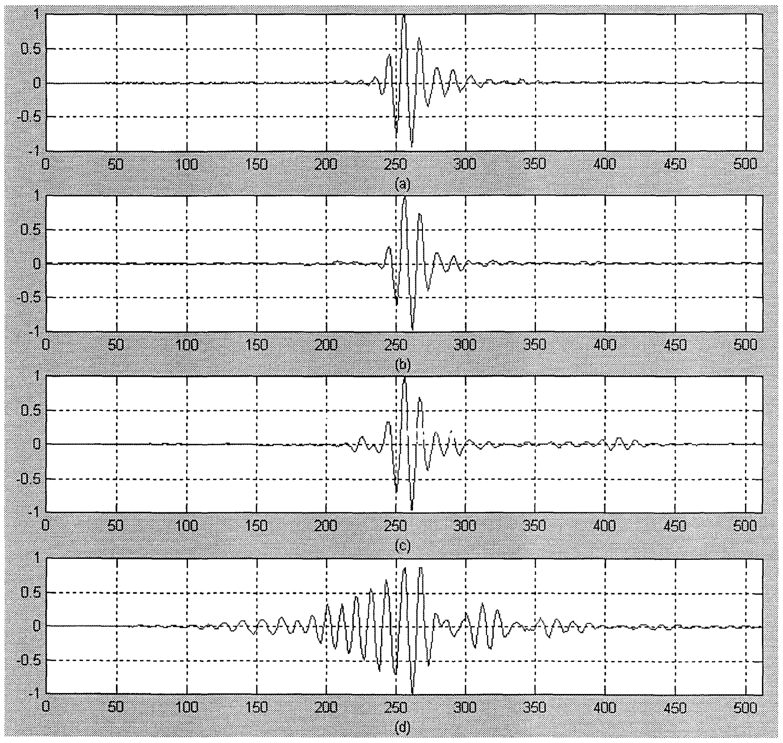

Figure 3. Typical A-scans (a) crack, (b) LOF, (c) slag, (d) porosity

Note from Table 1 that the first training dataset *SI* had instances only from crack and LOF, whereas *S2* and *S3* added instances from slag and porosities, respectively. LEARN++ was trained starting with *SI*. After generating 8 hypotheses using *SI*, LEARN++ was presented with instances from *S2*, removing all instances from *SI*. 27 hypotheses were generated using *S2*, after which LEARN++ was presented with instances only from *S3* to generate 43 additional hypotheses. The 800 test signals were never shown to the algorithm. The weak learner used to generate individual hypotheses was a single hidden layer MLP with 50 hidden layer nodes. The mean square error goals of all MLPs were preset to a value of 0.02 to prevent over fitting and to ensure a weak learning algorithm. Table 2 presents the classification results obtained by the algorithm.

As seen from Table 2, LEARN++ was able to correctly classify 99.2% of training instances in 57, but only 57% of the test instances, by combining 8 hypotheses. This is expected, since the 57 had instances only from two classes, whereas *TEST* had instances from all classes. After the next training session, using instances only from 52, the algorithm was able to correctly classify 89.2 % of instances in 57, and 86.5% of instances in 52. The performance on *TEST* dataset improved to 70.5%. Finally, after the final training session using instances from *S3*, the algorithm correctly classified 88.2%, 88.1% and 91.2% of instances in 57, 52, and 55, respectively. The classification performance on *TEST* dataset increased to 83.8%, demonstrating the incremental learning capability of the LEARN++ algorithm.

As a performance comparison, the same database was also used to train and test a single strong learner, a 149x40x12x4 two hidden layer MLP with an error goal of 0.001. The best test data classification performance of the strong learner has been around 75%, despite the fact that the strong learner was trained with the entire dataset at once.

Finally, LEARN++ was tested on the entire C-scan images, which consisted of tens of thousands of A-scans. On each C-scan, a previously identified rectangular region was selected and classified by LEARN++, creating a *classification image* of the selected rectangular region. Median filtering was then applied to the classification image to remove isolated pixels, producing the final classification image. Figures 4-6 illustrate examples of raw C-scans along with the rectangular region of interest, and their respective classification images. Table 3 presents the classification performance of LEARN ++ compared to that of the strong learner which was trained in one iteration using the entire dataset.

Table 2. Classification performance of Learn++ on ultrasonic weld inspection signals

Inc. Train→ ↓ Dataset	Training 1 (8)	Training 2 (27)	Training3 (43)
Si	99.2%	89.2%	88.2%
S2	-	86.5%	88.1%
SB	-	-	96.4%
TEST	57.0%	70.5%	83.8%

Table 3. Comparison of Learn++ and strong learner on C-scans of weld inspection data

	Number of C-scans (Training)	# of C-scans Missed (Training)	Classification Performance	# of C-scans (Validation)	# of C-scans Missed (Validation)	Classification Performance
Strong Learner	106	8	92.4 %	50	13	74.0%
Learn++	106	1	99.1%	50	11	78.0%

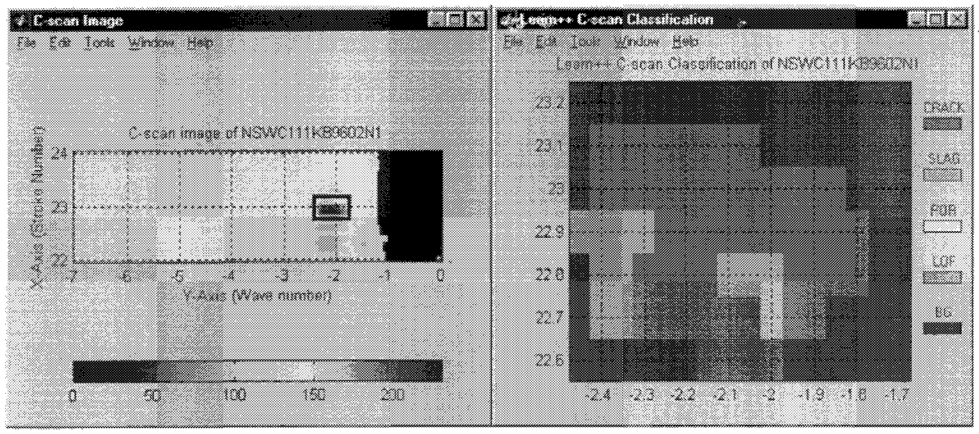
C-scan Image
File Edit Tools Window Help
C-scan image of NSWC111KB9602N1
X-Axis (Stroke Number)
Y-Axis (Wave number)
Learn++ C-scan Classification
Learn++ C-scan Classification of NSWC111KB9602N1
CRACK
SLAG
POR
LOF
BG

Figure 4. Original C-scan and Learn++ classification, correct class: Crack

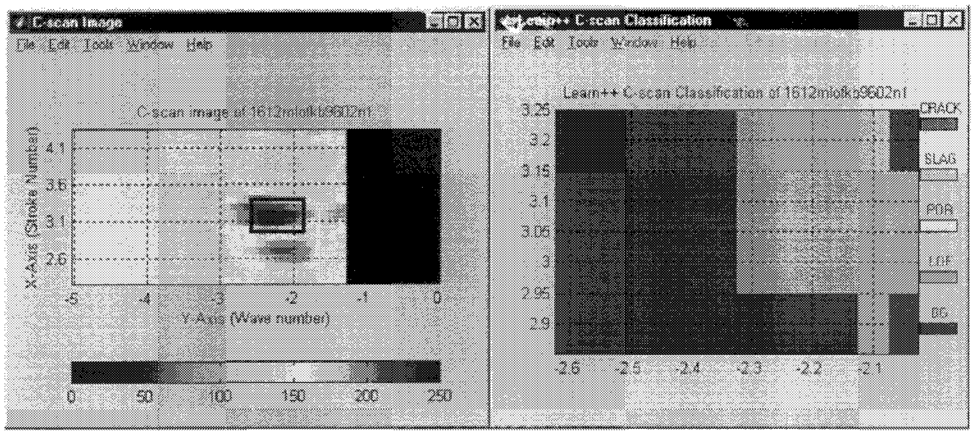
C-scan Image
File Edit Tools Window Help
C-scan image of 1612mlofkb9602n1
X-Axis (Stroke Number)
Y-Axis (Wave number)
Learn++ C-scan Classification
Learn++ C-scan Classification of 1612mlofkb9602n1
CRACK
SLAG
POR
LOF
BG

Figure 5. Original C-scan and Learn++ classification, correct class: LOF

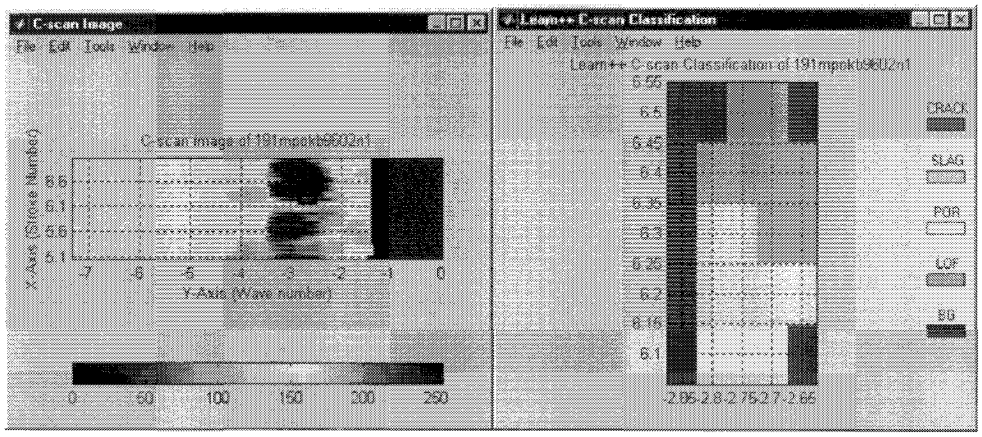
C-scan Image
File Edit Tools Window Help
C-scan image of 191mpokb9602n1
X-Axis (Stroke Number)
Y-Axis (Wave number)
Learn++ C-scan Classification
Learn++ C-scan Classification of 191mpokb9602n1
CRACK
SLAG
POR
LOF
BG

Figure 6. Original C-scan and Learn++ classification, correct class: Porosity

CONCLUSIONS AND FUTURE WORK

We have introduced LEARN++, an incremental learning algorithm for supervised neural networks, which employs an ensemble of networks for learning new data. The feasibility of the approach has been validated on an inherently difficult database of ultrasonic weld inspection signals. Note that LEARN++ was implemented using MLPs as weak learners, however, the algorithm itself is not dependent on the choice of a particular classification algorithm, and it should be able to work well on all supervised classifiers. Current work includes evaluating LEARN++ using other learning algorithms.

LEARN++ has two key components. The first one is the selection of the subsequent training dataset (the distribution update rule). This scheme can be improved to allow faster learning and reduced computational complexity. The second key component is the procedure by which hypotheses are combined. LEARN++ uses weighted majority voting, however, various other schemes can also be used. For example, the weight of a hypothesis can be learned using a subsequent learner, rather than estimating it from the performance of that hypothesis. Work is currently underway to address these issues.

The only drawback of LEARN++ is the computational burden due to the overhead added by computing multiple hypotheses, and saving all classifier parameters for these hypotheses. Since each classifier is a weak learner, it has fewer parameters than its strong counterpart, and therefore it can be trained much faster. Although total training time for LEARN++ is similar to or less than that of a strong learner, its space complexity is significantly higher. However, due to increased storage capacities that have recently become available, this drawback is considered to be insignificant.

REFERENCES

1. Nawapak, E.A., Udpa, L., Chao, J., "Morphological Processing for Crack Detection in Eddy Current Images of Jet Engine Disks," in *Review of Progress in Quantitative Nondestructive Evaluation*, edited by Thompson, D. O., and Chimenti, D.E., Plenum Press, New York, 1999, pp. 751-758.
2. Polikar, R., Udpa, L., Udpa, S.S., and Taylor, T., *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control* **45**, 614-625, (1998).
3. Ramuhalli, P., Udpa, L., Udpa, S.S., *Materials Evaluation* **58**, 65-69, (2000).
4. Polikar, R., *Algorithms for enhancing pattern separability, optimum feature selection and incremental learning with applications to gas sensing electronic nose systems*, Ph.D. dissertation, Iowa State University, 2000, Chapter 6.
5. Freund, Y., and Schapire, R., *Computer and System Sciences* **57**, 119-139 (1997).
6. Littlestone, N., and Warmuth, M., *Information and Computation* **108**, 212-261, (1994).
7. Wolpert, D.H., *Neural Networks* **5**, 241-259, (1992).
8. Kittler, J., Hatef, M., Duin, R.P., and Matas, J., *IEEE Transactions on Pattern Analysis and Machine Intelligence,* **20**, 226-239, (1998).
9. Rangarajan, S., Jalote, P., and Tripathi, S., *IEEE Trans. Software Engineering,*. **19**, 698-706, (1993).
10. Ji, C., and Ma, S., *IEEE Transactions on Neural Networks,* **8**, 32-42, (1997).
11. Ji, C., and Ma, S., *IEEE Proceedings,* **87**, 1519-1535, (1999).
12. Dietterich, T.G., *AI Magazine*, **18,** 97-136, (1997).