Ensemble of Classifiers Based Incremental Learning with Dynamic Voting Weight Update

Robi Polikar, Stefan Krause and Lyndsay Burd
Electrical and Computer Engineering, Rowan University,
136 Rowan Hall, Glassboro, NJ 08028, USA.

Abstract – An incremental learning algorithm based on weighted majority voting of an ensemble of classifiers is introduced for supervised neural networks, where the voting weights are updated dynamically based on the current test input of unknown class. The algorithm's dynamic voting weight update feature is an enhancement to our previously introduced incremental learning algorithm, Learn++. The algorithm is capable of incrementally learning new information from additional datasets that may later become available, even when the new datasets include instances from additional classes that were not previously seen. Furthermore, the algorithm retains formerly acquired knowledge without requiring access to datasets used earlier, attaining a delicate balance on the stability-plasticity dilemma. The algorithm creates additional ensembles of classifiers based on an iteratively updated distribution function on the training data that favors training with increasingly difficult to learn, previously not learned and/or unseen instances. The final classification is made by weighted majority voting of all classifier outputs in the ensemble, where the voting weights are determined dynamically during actual testing, based on the estimated performance of each classifier on the current test data instance. We present the algorithm in its entirety, as well as its promising simulation results on two real world applications.

I. INTRODUCTION

A. Incremental Learning

As most researchers in machine learning are painfully aware, the generalization performance of any learning algorithm is acutely contingent upon the availability of an adequate and representative training dataset. Often times, however, acquisition of such data is tedious, time consuming and expensive. Therefore, it is not unusual for such data to become available in batches over a period of time. Furthermore, it is also not unusual for data belonging to different classes to be acquired in separate data acquisition episodes. Depending on the exact nature of the application, waiting for the entire data to become available can be ineffective, financially uneconomical, technically improvident, or even unfeasible – particularly if the exact nature of future data is unknown. Under such scenarios, it would be more effective to be able to start training a classifier with the existing data, and then incrementally update this classifier to accommodate new data without, of course, compromising classification performance on previously seen data.

Since most of the commonly used classifiers, including the ubiquitous multilayer perceptron (MLP) and the radial basis function (RBF) networks are unable to accommodate such incremental learning, the practical approach has traditionally been discarding the existing classifier and starting from scratch by combining all data accumulated thus far every time a new dataset becomes available. This approach results in loss of all previously learned knowledge, a phenomenon known as *catastrophic forgetting*. Furthermore, the combination of old and new datasets is not even always possible if previous datasets are lost, discarded, corrupted, inaccessible, or otherwise unavailable.

Incremental learning of new information without forgetting what is previously learned raises the so-called *stability – plasticity dilemma* [1]: some information may have to be lost to learn new information, as learning new patterns will tend to overwrite formerly acquired knowledge. Thus, a stable classifier can preserve existing knowledge, but cannot accommodate new information, whereas a plastic classifier can learn new information, but cannot retain prior knowledge. The issue at hand is then, if, when and how much should one be sacrificed for the other to achieve a meaningful balance.

Various definitions and interpretations of incremental learning can be found in literature. A representative, yet certainly not exhaustive, list of references can be found in [2].

For the purposes of this work, an algorithm possesses incremental learning capabilities, if it meets the following criteria: (1) ability to acquire additional knowledge when new datasets are introduced, without requiring access to previously seen data; (2) ability to retain a meaningful portion of the previously learned information; and (3) ability to learn new classes if introduced by new data.

Algorithms referenced in [2], as well as many others are all capable of learning new information; though they satisfy the above-mentioned criteria only at varying degrees: they either require access to previously seen data, forget substantial amount of prior knowledge along the way, or cannot accommodate new classes. One prominent exception is the (fuzzy) ARTMAP algorithm [3] along with its many recent variations. However, it has long been known that ARTMAP is very sensitive to selection of its vigilance parameter, to noise levels in the training data and to the order in which the training data are presented to the algorithm. Furthermore, the algorithm often suffers from overfitting problems, if the vigilance parameter is not chosen appropriately. Various approaches have been and are being suggested to overcome these difficulties [4, 5, 6, 7, 8].

Recently we have suggested Learn++ as an alternate approach to the growing list of incremental learning algorithms [2,9]. Learn++, based on the weighted majority voting of an ensemble of classifiers, satisfies the above mentioned criteria. However, the weights for the majority voting are set during training and remain constant (hence static

0-7803-7898-9/03/$17.00 ©2003 IEEE

weights) after the training. In this paper, we present a modified version of the algorithm, where voting weights are updated dynamically by estimating which of the classifiers are likely to correctly classify any given test instance.

B. Ensemble of Classifiers

Learn++ takes advantage of the synergistic power of an ensemble of classifiers in learning a concept using the *divide and conquer* approach. The algorithm is in part inspired by the AdaBoost *(adaptive boosting)* algorithm [10], originally developed to improve the classification performance of weak classifiers. In essence, an *ensemble of weak classifiers* are trained using different distributions of training samples, whose outputs are then combined using the weighted majority-voting scheme [11] to obtain the final classification rule. The approach exploits the so-called *instability* of the weak classifiers, which allows the classifiers to construct sufficiently different decision boundaries for minor modifications in their training datasets, causing each classifier to make different errors on any given instance. A strategic combination of these classifiers then eliminates the individual errors, generating a strong classifier.

Using ensemble of classifiers has been well researched for improving classifier accuracy [12, 13, 14, 15, 16]; however, its potential for addressing the incremental learning problem has been mostly unexplored. Learn++ was developed in response to recognizing the potential feasibility of ensemble of classifiers in solving the incremental learning problem.

Learn++ was first introduced in [17, 18], as an incremental learning algorithm for MLP networks. More recently we showed that Learn++ is actually quite versatile, as it works with any supervised classification algorithm [9]. In its original form, Learn++ combines the classifiers using weighted majority voting, where the voting weights are determined by individual performances of the classifiers on their own training data. We realize that this is sub optimal, as a classifier that performs well on its own training data need not perform equally well on data coming from a different portion of the input space. In this paper, we first introduce the modified Learn++ algorithm, which uses a statistical-distance-metric based procedure for estimating which classifiers are likely to correctly classify a given unknown instance. Higher weights are then assigned to those classifiers estimated to perform well on the given instance.

II. LEARN++

Learn++ generates a set of classifiers (hypotheses) and combines them through weighted majority voting of the classes predicted by the individual hypotheses. The hypotheses are generated by training a weak classifier, using instances drawn from iteratively updated distributions of the training database. The distribution update rule used by Learn++ is designed to accommodate additional datasets, in particular those that introduce previously unseen classes. Each classifier is trained using a subset of examples drawn from a weighted distribution that gives higher weights to examples misclassified by the previous *ensemble*. The pseudocode of the algorithm is provided in Fig. 1.

For each database $\mathcal{D}_k$, $k=1,\ldots,K$ that becomes available to the algorithm, the inputs to Learn++ are (1) labeled training data $S_k = \{(\mathbf{x}_i, \mathbf{y}_i) \mid i = 1,\cdots,m_k\}$ where $\mathbf{x}_i$ and $\mathbf{y}_i$ are training instances and their correct classes, respectively ; (2) a weak-learning algorithm **BaseClassifier**; and (3) an integer T_k, the maximum number of classifiers to be generated. For brevity we drop the subscript k from all other variables. BaseClassifier can be any supervised algorithm that achieves at least 50% correct classification on S_k after being trained on a subset of S_k. This ensures that the classifier is sufficiently weak, yet strong enough to ensure at least a meaningful classification performance. We also note that using weak classifiers has the additional advantage of rapid training and overfitting avoidance, since they only generate a gross approximation of the underlying decision boundary.

At each iteration t, Learn++ first initializes a distribution $\mathbf{D}_t$, by normalizing a set of weights, $\mathbf{w}_t$, assigned to instances based on their individual classification by the current ensemble (step 1)

$$\mathbf{D}_t = \mathbf{w}_t \Big/ \sum_{i=1}^{m} w_t(i) \,. \tag{1}$$

Learn++ then divides S_k into two mutually exclusive subsets by drawing a training subset TR_t and a test subset TE_t according to $\mathbf{D}_t$ (step 2). Unless there is prior reason to choose otherwise, $\mathbf{D}_t$ is initially set to be uniform, giving equal probability to each instance to be selected into TR_1. Learn++ then calls BaseClassifier to generate hypothesis h_t (step 3). The error of h_t is computed on $S_k = TR_t \bigcup TE_t$ by adding the distribution weights of misclassified instances (step 4)

$$\varepsilon_t = \sum_{i:h_t(\mathbf{x}_i)\neq \mathbf{y}_i} D_t(i) = \sum_{i=1}^{m_k} D_t(i)\left[\!\left[h_t(\mathbf{x}_i) \neq \mathbf{y}_i \right]\!\right] \tag{2}$$

where $[\![\bullet]\!]$ is 1 if the predicate is true, and 0 otherwise. If $\varepsilon_t > ½$, current h_t is discarded and a new h_t is generated from a fresh set of TR_t and TE_t. If $\varepsilon_t < ½$, then normalized error β_t is computed as

$$\beta_t = \varepsilon_t / (1-\varepsilon_t)\,, \quad 0 < \beta_t < 1\,. \tag{3}$$

All hypotheses generated during the previous t iterations are then combined using weighted majority voting (step 5), to construct the *composite hypothesis* H_t

$$H_t = \arg\max_{\mathbf{y}\in Y} \sum_{t:h_t(\mathbf{x})=\mathbf{y}} \log\frac{1}{\beta_t}\,. \tag{4}$$

H_t decides on the winning class that receives the highest total vote. In the original Learn++ algorithm, the voting weights were determined based on the normalized errors β_t: hypotheses with lower normalized errors are given larger weights, so that the classes predicted by a hypothesis with a proven record are weighted more heavily. The log function is used to control the explosive effect of very low β_t values associated with classifiers that perform well during training.

Algorithm Learn++

Input: For each dataset drawn from $\mathcal{D}_k$ $k=1,2,\ldots,K$

- Sequence of m_k examples $S_k = \{(\mathbf{x}_i, \mathbf{y}_i) \mid i = 1,\cdots,m_k\}$
- Weak learning algorithm **BaseClassifier**.
- Integer T_k, specifying the number of iterations.

Do for each $k=1,2,\ldots,K$:

Initialize $w_1(i) = D_1(i) = 1/m,\ \forall i,\ i = 1,2,\cdots,m$

Do for $t = 1,2,\ldots,T_k$:

1. Set $D_t = \mathbf{w}_t \Big/ \sum_{i=1}^{m} w_t(i)$ so that D_t is a distribution.
2. Draw training TR_t and testing TE_t subsets from D_t.
3. Call **BaseClassifier** to be trained with TR_t.
4. Obtain a hypothesis h_t : X → Y, and calculate the error of h_t : $\varepsilon_t = \sum_{i:h_t(\mathbf{x}_i)\neq \mathbf{y}_i} D_t(i)$ on $TR_t + TE_t$.

   If $\varepsilon_t > ½$, discard h_t and go to step 2. Otherwise, compute normalized error as $\beta_t = \varepsilon_t / (1-\varepsilon_t)$.
5. Call dynamically weighted majority voting to obtain composite hypothesis $H_t = \arg\max_{\mathbf{y}\in Y} \sum_{t:h_t(\mathbf{x})=\mathbf{y}} DW_t(x)$
6. Compute the error of the composite hypothesis

$$E_t = \sum_{i:H_t(\mathbf{x}_i)\neq \mathbf{y}_i} D_t(i) = \sum_{i=1}^{m_k} D_t(i)\left[\left| H_t(\mathbf{x}_i) \neq \mathbf{y}_i \right|\right]$$

7. Set $B_t = E_t/(1-E_t)$, and update the weights:

$$w_{t+1}(i) = w_t(i) \times \begin{cases} B_t, & \text{if } H_t(\mathbf{x}_i) = \mathbf{y}_i \\ 1, & \text{otherwise} \end{cases} = w_t(i) \times B_t^{1-\left[\left|H_t(\mathbf{x}_i)\neq \mathbf{y}_i\right|\right]}$$

Call Dynamically weighted majority voting and **Output** the final hypothesis:

$$H_{final}(\mathbf{x}) = \arg\max_{\mathbf{y}\in Y} \sum_{k=1}^{K} \sum_{t:h_t(\mathbf{x})=\mathbf{y}} DW_t(\mathbf{x})$$

Fig 1. Algorithm Learn++

While this rule makes intuitive sense, and in fact performed remarkably well in a number of simulations [9], it is nevertheless sub optimal. This is because the weights are determined – and fixed prior to testing – based on individual performances of hypotheses on their own training data subset. A rule that dynamically estimates which hypotheses are likely to correctly classify an unlabeled instance, to give higher voting weights to those hypotheses would be more optimal. We therefore change the composite hypothesis expression as

$$H_t = \arg\max_{\mathbf{y}\in Y} \sum_{t:h_t(\mathbf{x})=\mathbf{y}} DW_t(\mathbf{x}) \tag{5}$$

where $DW_t(\mathbf{x})$ is the instance-specific dynamic weight assigned to instance $\mathbf{x}$ by the hypothesis h_t. Dynamic weights are determined using a Mahalanobis-distance-based estimated likelihood of h_t for correctly classifying $\mathbf{x}$, as described below.

The composite error E_t made by H_t is then computed as the sum of distribution weights of instances misclassified by H_t (step 6)

$$E_t = \sum_{i:H_t(\mathbf{x}_i)\neq \mathbf{y}_i} D_t(i) = \sum_{i=1}^{m_k} D_t(i)\left[\left| H_t(\mathbf{x}_i) \neq \mathbf{y}_i \right|\right]. \tag{6}$$

The composite normalized error is similarly computed as

$$B_t = E_t / 1 - E_t, 0 < B_t < 1. \tag{7}$$

The weights $w_t(i)$ are then updated, for computing the next distribution $\boldsymbol{D}_{t+1}$, which in turn is used in selecting the next training and testing subsets, TR_{t+1} and TE_{t+1}, respectively (step 7)

$$w_{t+1}(i) = w_t(i) \times \begin{cases} B_t, & \text{if } H_t(\mathbf{x}_i) = \mathbf{y}_i \\ 1, & \text{otherwise} \end{cases} = w_t(i) \times B_t^{1-\left[\left|H_t(\mathbf{x}_i)\neq \mathbf{y}_i\right|\right]}. \tag{8}$$

This rule reduces the weights of those instances correctly classified by the composite hypothesis H_t by a factor of B_t (since $0 < B_t < 1$), whereas it leaves the weights of misclassified instances unchanged. After normalization (in step 1 of iteration $t+1$), the probability of correctly classified instances being chosen into TR_{t+1} is reduced, while those of misclassified ones are effectively increased. Therefore, the algorithm focuses on instances that are difficult to classify, or instances that have not yet been properly learned. This approach allows incremental learning by concentrating on newly introduced instances, particularly those coming from previously unseen classes, as these are precisely those instances that have not been learned yet.

We emphasize the introduction of the composite hypothesis H_t by Learn++, which along with the weight update rule, uniquely allows Learn++ to learn new classes. It is empirically observed that the procedure fails to learn new classes, if instead the weight update rule were based on the performance of h_t only (as AdaBoost does). The weight update rule based on composite hypothesis performance allows Learn++ to focus on those instances that have not been learned by the *current ensemble*, rather then the *previous hypothesis*.

After T_k hypotheses are generated for each database $\mathcal{D}_k$, the final hypothesis H_{final} is obtained by combining all hypotheses that have been generated thus far using the dynamically weighted majority-voting rule choosing the class that receives the highest total vote among all classifiers,

$$H_{final}(\mathbf{x}) = \arg\max_{\mathbf{y}\in Y} \sum_{k=1}^{K} \sum_{t:h_t(\mathbf{x})=\mathbf{y}} DW_t(\mathbf{x}). \tag{9}$$

To determine the dynamic voting weights, we use the Mahalanobis distance, to compute the distance of the unknown instance to the datasets used to train individual classifiers. Classifiers trained with datasets closer to the unknown instance are then given larger weights. Note that this approach does not require the training data to be saved, but only the mean and covariance matrices, which are typically much smaller than the original data. In Learn++, we first define TR_{tc} as a subset of TR_t, the training data used during the t^{th} iteration, where TR_{tc} includes only those instances of TR_t that belong to class c, that is,

$$TR_{tc} = \{\mathbf{x}_i \mid \mathbf{x}_i \in TR_t \ \&\ \mathbf{y}_i = c\} \ni TR_t = \bigcup_{c=1}^{C} TR_{tc} \qquad (10)$$

where C is the total number of classes. The class specific Mahalanobis distance from an unknown instance $\mathbf{x}$ to TR_{tc}, is then computed as

$$M_{tc}(x) = (\mathbf{x} - \mathbf{m}_{tc})^T \mathbf{C}_{tc}^{-1} (\mathbf{x} - \mathbf{m}_{tc}), \quad c = 1,2,\cdots,C \qquad (11)$$

where $\mathbf{m}_{tc}$ is the mean and $\mathbf{C}_{tc}$ is the covariance matrix of TR_{tc}. For any instance $\mathbf{x}$, the Mahalanobis distance based dynamic weight of the t^{th} hypothesis can then be obtained as

$$DW_t(\mathbf{x}) = \frac{1}{\min(M_{tc}(\mathbf{x}))}, \quad c = 1,2,\cdots,C. \ t = 1,\cdots,T \qquad (12)$$

where T is the total number of hypotheses generated.

The Mahalanobis distance metric implicitly assumes that the data is drawn from a Gaussian distribution, which in general is not the case. However, this metric provided promising results demonstrating its effectiveness. Other distance metrics that do not make this assumption will also be evaluated.

III. LEARN++ SIMULATION RESULTS

The original Learn++ algorithm using the static weighted majority voting has been tested on a number of benchmark and other real world databases, whose results are provided in [2,9]. In this paper, we present simulation results of using Learn++ with dynamically updated weighted majority voting on two real-world incremental learning problems that introduce new classes with additional data.

A. Ultrasonic Weld Inspection (UWI) Database

This rather challenging database was obtained by ultrasonic scanning of welding regions of various stainless or carbon steel structures. The welding regions, known as heat-affected zones, are highly susceptible to growing a variety of defects, including potentially dangerous cracks. The discontinuities within the material, such as the air gaps due to cracks, cause the ultrasonic wave to be reflected back and received by the transducer. The reflected ultrasonic wave, also called the A-scan, serves as the signature pattern of the discontinuity, which is then analyzed to determine its type. This analysis, however, is hampered by the presence of other types of discontinuities, such as porosity (POR), slag and lack of fusion (LOF), all of which generate very similar A-scans, creating highly overlapping patterns in the feature space. Representative A-scans are shown in Fig.2

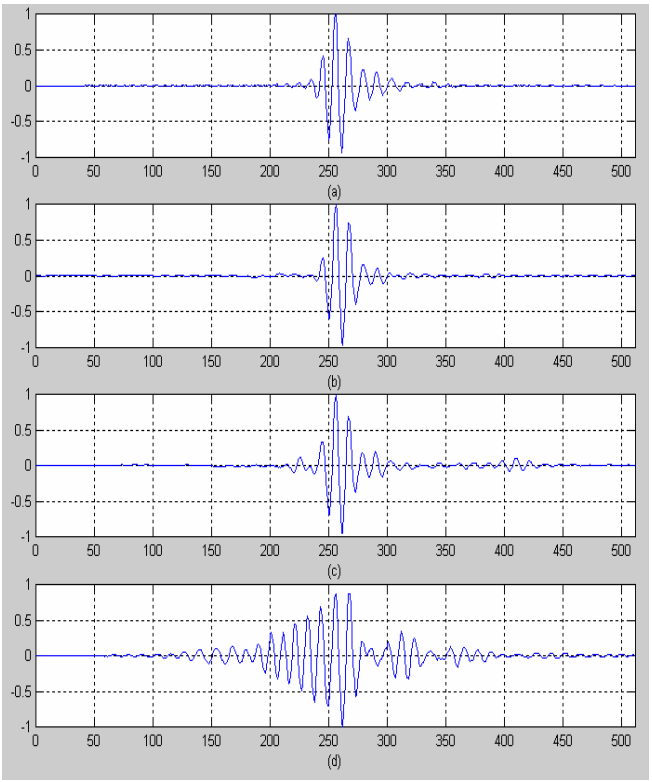

Fig. 2 Representative normalized A-scans of (a) crack, (b) Lack of fusion, (c) slag, (d) porosity

This four-class database was divided into three training datasets, $S_1 \sim S_3$, and a validation set, *TEST*. Table 1 presents the data distribution in each dataset. We note that each additional database introduced a new class: in particular, S_1 had instances only from crack and LOF, S_2 introduced slag instances and S_3 introduced porosity instances. *TEST* dataset included instances from all classes.

Only S_k was used during the k^{th} training session TS_k. Therefore, previously used data were not made available to Learn++ in future training sessions. Table 2 summarizes the training and generalization performances of the algorithm after each training session. The classification performances on training and testing datasets are shown in rows labeled by S_1, S_2, S_3, and *TEST* as obtained after each training session, TS_k. The numbers in parentheses indicate the number of weak classifiers generated in each training session. Classifier generation continued until the generalization performance on the *TEST* dataset, shown in the last row, reached a steady saturation value. 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 sufficiently weak learning. We note that any neural network can be turned into a weak learning algorithm by selecting its number of hidden layers and the number of hidden layer nodes small, and the error goal high, with respect to the complexity of the problem.

TABLE 1. DATA DISTRIBUTION FOR THE UWI DATABASE

↓Dataset	LOF	SLAG	CRACK	POR
S_1	300	300	0	0
S_2	150	300	150	0
S_3	200	250	250	300
TEST	**300**	**300**	**200**	**100**

TABLE 2. LEARN++ PERFORMANCE ON THE UWI DATABASE

↓ Dataset	TS_1 (8)	TS_2 (27)	TS_3 (43)
S_1	99.2%	89.2%	88.2%
S_2	---	86.5%	88.1%
S_3	---	---	96.4%
TEST	**57.0%**	**70.5%**	**83.8%**

Several observations can be made from Table 2:

(i) The generalization performances on *TEST* dataset steadily increase, indicating that Learn++ is indeed able to learn new information, as well as new classes as they become available;

(ii) In general, larger numbers of classifiers are required for incremental learning of new classes.

(iii) There is an occasional decline on training data performances indicating some loss, albeit minor, of previously acquired knowledge as new information is acquired. This is expected due to stability-plasticity dilemma;

(iv) Near 50% generalization performance after TS_1 makes intuitive sense, since at that time the algorithm had only seen two of the four classes that appear in the *TEST* data. The performance gradually increases proportional to the ratio of the classes seen, as they become available.

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 instances from all classes.

B. Volatile Organic Compound (VOC)Database

This database was obtained from responses of six quartz crystal microbalances (QCMs) to various concentrations of five volatile organic compounds (VOCs), including ethanol (ET), xylene (XL), octane (OC), toluene (TL), and trichloroethelene (TCE). When QCMs are exposed to VOCs, the molecular mass deposited on their crystal surface alters their resonant frequency, which can be measured using a frequency counter or a network analyzer. By using an array of QCMs, each coated with a different polymer sensitive to specific VOCs, the collective response of the array can be used as a signature pattern of the VOC. However, QCMs have very limited selectivity, making the identification a challenging task. Representative patters of sensor responses for each VOC are shown in Fig. 3.

Similar to the UWI database, the VOC identification database was divided into three training datasets $S_1 \sim S_3$ and one validation dataset, *TEST*. The data distribution is shown in Table 3, which was specifically biased towards the new class: database S_1 had instances from ET, OC and TL, S_2 added instances mainly from TCE (and very few from the previous three), and S_3 added instances from XL (and very few from the previous four). *TEST* set included instances from all classes. The base classifier used for this database was also a MLP type neural network with a 6x30x5 architecture, and an error goal of 0.05. Table 4 presents the training and generalization performances of the algorithm on this database.

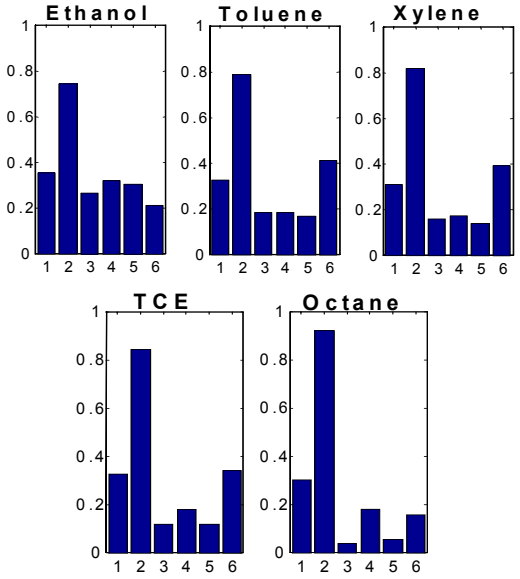

Fig. 3 Representative QCM sensor responses to five VOCs

TABLE 3.DATA DISTRIBUTION FOR THE VOC DATABASE

↓Dataset	ET	OC	TL	TCE	XL
S1	20	20	40	0	0
S2	10	10	10	25	0
S3	10	10	10	15	40
TEST	**24**	**24**	**52**	**24**	**40**

TABLE 4. LEARN++ PERFORMANCE ON THE VOC DATABASE

↓Dataset	TS_1 (7)	TS_2 (10)	TS_3 (16)
S_1	100%	97.5%	92.5%
S_2	---	96.4%	96.4%
S_3	---	---	92.9%
TEST	**57.3%**	**67.1%**	**89.6%**

Performance figures listed above follow a similar trend to those of the UWI database. The steady increase on the generalization performance indicates that Learn++ was able to incrementally learn additional information provided by the new classes.

We have evaluated Learn++ numerous times for each database with slightly different algorithm parameters, such as the order of introduction of the classes, base classifier architecture and/or error goal, the percentage of training data used as training and testing subsets, etc. and even to a great extend the choice of base classifier. In all cases, the algorithm seemed to be remarkably resistant to such changes.

IV. DISCUSSIONS & CONCLUSIONS

In this paper, we introduced a new version of the Learn++ algorithm. Learn++ essentially adds the incremental learn-

ing capability to any supervised neural network that normally does not possess this property. As demonstrated by the results presented in this and our previous papers, Learn++ is indeed able to learn new information provided by additional databases, even when the new classes are introduced by the consecutive datasets. Learn++ takes advantage of the synergistic expressive power of an ensemble of weak classifiers, where each classifier is trained with a training data subset drawn from a strategically updated distribution of the training data. Individual classifiers are then combined using the weighted majority-voting rule to obtain the final classifier. In this paper, we proposed an alternate weighted majority voting strategy, where the voting weights are determined dynamically for each instance, based on the estimated likelihood of the hypotheses to correctly classify that instance. The intuitive idea behind this approach is that those classifiers trained with a dataset that included nearby instances– in the Mahalanobis distance sense – to the unknown instance are more likely to correctly classify the unknown instance.

It can be argued that the classification decision is already being made by the choice of the class providing the minimum Mahalanobis distance, since if instance **x** belongs to a particular class, and instances from that class have been used in the current dataset, then the Mahalanobis distance between **x** and TR_{tc} is likely to be minimum among all others. This is indeed true for datasets with non-overlapping classes with noise-free well-behaving distributions. In practice, however, this is rarely the case, and a decision based on the Mahalanobis distance only invariably achieves poor classification performances on challenging datasets, such as the UWI and VOC datasets. We emphasize that the Mahalanobis distance is not used directly for making a classification decision, but rather to assign a weight to competing hypotheses.

Through out the simulations, a number of additional observations were made, not readily apparent from the tables. In particular, we have tested the sensitivity of Learn++ to the order of presentation of the data, as well as to minor changes in its parameters. We found out that the classification performance of Learn++ was virtually the same, regardless of the order in which databases (and therefore the classes) were presented to the algorithm. Furthermore, the algorithm was considerably robust to changes in its internal parameters, such as the network size, error goal, the number of hypotheses generated, and even the type of base classifier.

Finally, unlike most other supervised classifiers, Learn++ does not suffer from catastrophic forgetting, since previously generated classifiers are retained. Due to the stability – plasticity dilemma, some knowledge is indeed forgotten while new information is being learned; however, this appears to be insignificant, as indicated by the steady improvement in the generalization performance.

One might also wonder what generalization performance could be achieved if the entire database were available for strong learning. Training a strong classifier using the entire training database, we obtain performances in the mid 70% to high 80% range, similar to, or even slightly worse than those of Learn++. This rather satisfying result further indicates the feasibility of Learn++ as an alternative to other incremental learning algorithms, as well as to non-incremental strong classifiers.

ACKNOWLEDGEMENT

This material is based upon work supported by the National Science Foundation under Grant No. ECS-0239090.

REFERENCES

[1] S. Grossberg, "Nonlinear neural networks: principles, mechanisms and architectures," *Neural Networks*, vol. 1, no. 1, pp. 17-61, 1988.

[2] R. Polikar, L. Udpa L, S. Udpa, and V. Honavar, "Learn++: An incremental learning algorithm for supervised neural networks," *IEEE Transactions on System, Man and Cybernetics (C), Special Issue on Knowledge Management*, vol. 31, no. 4, pp. 497-508, 2001.

[3] G.A. Carpenter, S. Grossberg, N. Markuzon, J.H. Reynolds, and D.B. Rosen, "Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps," *IEEE Trans. on Neural Networks*, vol. 3, no. 5, pp. 698-713, 1992.

[4] J.R. Williamson, "Gaussian ARTMAP: a neural network for fast incremental learning of noisy multidimensional maps," *Neural Networks*, vol. 9, no. 5, pp. 881-897, 1996.

[5] C.P. Lim and R.F. Harrison, "An incremental adaptive network for on-line supervised learning and probability estimation," *Neural Networks*, vol. 10, no. 5, pp. 925-939, 1997.

[6] F. H. Hamker, "Life-long learning cell structures – continuously learning without catastrophic interference," *Neural Networks*, vol. 14, no. 4-5, pp. 551-573, 2000.

[7] G.C. Anagnostopoulos and M. Georgiopoulos, "Ellipsoid ART and ARTMAP for incremental clustering and classification," *Int. Joint Conf. on Neural Networks (IJCNN 2001)*, vol. 2, pp. 1221-1226, 2001.

[8] E. Gomez-Sanchez, Y.A. Dimitriadis, J.M. Cano-Izquierdo, J. Lopez-Coronado, Safe-μARTMAP: A new solution for reducing category proliferation in Fuzzy ARTMAP, *Proc. of Int. Joint Conf. on Neural Networks (IJCNN 2001)*, vol.2, pp. 1197-1202, 2001.

[9] R. Polikar, J. Byorick, S. Krause, A. Marino, and M. Moreton, "Learn++: A classifier independent incremental learning algorithm for supervised neural networks," *Proc. of Int. Joint Conference on Neural Networks (IJCNN 2002),* vol.2, pp. 1742-1747, Honolulu, HI, 2002.

[10] Y. Freund and R. Schapire, "A decision theoretic generalization of on-line learning and an application to boosting," *Computer and System Sciences*, vol. 57, no. 1, pp. 119-139, 1997.

[11] N. Littlestone and M. Warmuth, "Weighted majority algorithm, " *Information and Computation*, vol. 108, pp. 212-261, 1994.

[12]L.K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp. 993-1001, 1990.

[13] M.I. Jordan and R.A. Jacobs, " Hierarchical mixtures of experts and the EM algorithm," *Proc. of Int. Joint Conf. on Neural Networks (IJCNN 1993),* pp. 1339-1344, 1993.

[14] J. Kittler, M. Hatef, R.P. Duin, J. Matas, "On combining classifiers," *IEEE Trans. on Pattern Analysis and Machine Intelligence,* vol. 20, no.3, pp. 226-239, 1998.

[15] T.G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting and randomization," *Machine Learning*, vol. 40, no. 2, pp. 1-19, 2000.

[16] L.I. Kuncheva, "A theoretical study on six classifier fusion starategies, " *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 2, pp. 281-286, 2002.

[17] R. Polikar, "Algorithms for enhancing pattern separability, feature selection and incremental learning with applications to gas sensing electronic nose systems," *Ph.D. dissertation*, Iowa State University, Ames, IA, 2000.

[18] R. Polikar, L. Udpa, S. Udpa, V. Honavar, "Learn++: An incremental learning algorithm for multilayer neural networks," Proc. of IEEE Int. Conf. on Acoustics, Speech and Signal Proc., vol. 6, pp. 3414-3417, 2000.