LEARN++: AN INCREMENTAL LEARNING ALGORITHM FOR MULTILAYER PERCEPTRON NETWORKS

R. Polikar[*], *L.Udpa*[*], *S.S. Udpa*[*], *V. Honavar*[**]
[*]Dept. of Electrical and Computer Engineering, [**]Dept. of Computer Science
Iowa State University
Ames, Iowa 50011, USA

ABSTRACT

We introduce a supervised learning algorithm that gives neural network classification algorithms the capability of learning incrementally from new data without forgetting what has been learned in earlier training sessions. Schapire's boosting algorithm, originally intended for improving the accuracy of weak learners, has been modified to be used in an incremental learning setting. The algorithm is based on generating a number of hypotheses using different distributions of the training data and combining these hypotheses using a weighted majority voting. This scheme allows the classifier previously trained with a training database, to learn from new data when the original data is no longer available, even when new classes are introduced. Initial results on incremental training of multilayer perceptron networks on synthetic as well as real-world data are presented in this paper.

1. INTRODUCTION

Learning from new data without forgetting prior knowledge is known as incremental learning, and it is an issue of paramount importance in automated data analysis systems. The problem arises from the fact that most existing classification algorithms do not allow incremental learning. Traditionally, when new data become available, these algorithms are reinitialized and retrained using a combination of old and new data, resulting in loss of all previous learning. Furthermore, the original data may not be available when new data arrives, making incremental learning impossible for such algorithms. This phenomenon is known as "*catastrophic forgetting*", and it is a common problem of many automated signal classification algorithms, including the multilayer perceptron (MLP), radial basis function, probabilistic, wavelet, and Kohonen networks. Therefore, a general approach that allows classification algorithms to learn incrementally would be of very beneficial, in particular to the signal processing community involved in automated classification and characterization of signals.

Incremental learning has been a popular subject of study in machine learning, and several versions of this problem have been addressed in the literature [1]. In one extreme case, incremental learning is trivialized by allowing retraining with old data, without adding new classes. On the other extreme end, an incremental learning algorithm is expected 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. Algorithms that are currently available for incremental learning, such as ARTMAP, typically fall somewhere in the middle of this spectrum. It is the opinion of the authors that for an algorithm to be considered as a truly incremental learning algorithm, it should not require access to the old data.

In this paper, we present LEARN++, an algorithm for MLP type neural networks (NN) allowing incremental learning from new data, which may or may not include new classes. We assume that database(s) with which the original network was trained is no longer available.

LEARN++ is inspired by Schapire's *adaptive boosting* algorithm, originally proposed for improving the accuracy of weak learning algorithms. In "Strength of weak learning" [2], Schapire showed that for a two class problem, a *weak learner* that almost always achieves high errors can be converted to a *strong learner* that almost always achieves arbitrarily low errors using *boosting*. Boosting is based on a majority voting of hypotheses (classification rules) generated by the weak learner for various distributions of the training data. Independently, Littlestone et al developed the *weighted majority algorithm*, which assigns weights to different hypotheses based on an error criterion, to construct a compound hypothesis which was proved to perform better than any of the individual hypotheses [3]. They also showed that the error of the compound hypothesis is closely linked to the mistake bound of the best hypothesis. Schapire and Freund later developed AdaBoost.M1 extending boosting to multiclass learning problems [4].

The original AdaBoost.M1 and modifications made to this algorithm for obtaining incremental learning capability are discussed in Section 2, followed by simulation results on synthetic and real world data in Section 3. Conclusions and discussion are presented in Section 4 along with directions for future work.

2. INCREMENTAL LEARNING USING LEARN++

2.1 Boosting Performance with AdaBoost.M1

AdaBoost.M1 was developed to boost the performance of a weak learner classifier by generating various weak classification hypotheses and combining them through weighted majority voting of the classes predicted by the individual hypotheses. These hypotheses are obtained by retraining the classifier using different distributions of the training database.

Inputs to AdaBoost.M1 are a sequence of labeled examples (training data, S), a weak learning algorithm, **WeakLearn**, and an integer T that specifies the number of (iterations) hypotheses

0-7803-6293-4/00/$10.00©2000 IEEE

to be generated by **WeakLearn.** AdaBoost.M1 iteratively updates the distribution of S by assigning appropriate weights to each instance such that the weak learner, which is trained with a subset training data drawn from this distribution, is forced to focus on increasingly harder instances. Thus the weak learner is challenged to learn the difficult parts of the instance space. A subset training data is drawn for each weak hypothesis.

At iteration t, AdaBoost.M1 provides **WeakLearn** with a subset training data drawn according to distribution D_t from the original training data $S=[(x_1,y_1),(x_2, y_2),\ldots,(x_m,y_m)]$, where x_i are training instances and yi are the correct labels for $i=0,1,\ldots,m$ instances. **WeakLearn** then computes a hypothesis (classifier) h_t: X → Y, which should correctly classify a fraction of the training set with respect to D_t. That is, **WeakLearn**'s goal is to find a hypothesis h_t, which minimizes the training error

$$\varepsilon_t = \sum_{i:h_t(x_i)\neq y_i} D_t(i) \tag{1}$$

AdaBoost.M1 requires that $\varepsilon_t < ½$ for each h_t, that is, each hypothesis must obtain a minimum performance of 50%. The initial distribution D_1 is uniform over S, that is, $D_1(i) = 1/m$, $\forall i$. This gives equal probability to all instances in S to be drawn into subset training data. The distribution is updated by

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t, & \text{if } h_t(x_i) = y_i \\ 1, & \text{otherwise} \end{cases} \tag{2}$$

where $Z_t = \sum_i D_t(i)$ is a normalization constant chosen to ensure that D_{t+1} will be a distribution, and $\beta_t=\varepsilon_t / (1-\varepsilon_t)$. Essentially, *easy* instances that are correctly classified by h_t get lower probability, and *hard* instances that are misclassified get higher probability of being selected for the next subset training data. Thus, AdaBoost.M1 focuses on the examples that seem to be hardest for **WeakLearn** to learn.

At the end of T iterations, AdaBoost.M1 combines the weak hypotheses $h_1,\ldots,h_T$ into a single final hypothesis h_{final} by computing the weighted majority of the weak hypotheses as

$$h_{final}(x) = \arg\max_{y\in Y} \sum_{t:h_t(x)=y} \log(1/\beta_t) \tag{3}$$

For a given instance x, h_{final} outputs the label y that maximizes the sum of the weights of the weak hypotheses predicting that label. The weight of hypothesis h_t is defined to be log $(1/\beta_t)$ so that greater weight is given to a hypothesis with lower error.

Note that AdaBoost.M1 requires a weak learner that can achieve a minimum of 50% classification accuracy. For a binary class problem, this is the least restrictive requirement one could have. However, obtaining an error of ½ becomes increasingly difficult as the number of classes increase, since for a k class problem, the error for random guessing is $(k-1)/k$. Therefore, the choice of a weak learning algorithm with a classification performance of at least 50% may not be very easy. NN algorithms, however, can be used as weak learners, since they can be made weaker or stronger by modifying their parameters. For example, an MLP with larger number of nodes/layers and a smaller error goal is stronger than the one with smaller number of nodes and a higher error goal. It should be noted that using strong learners that achieve high classification performance on a particular training data are not recommended for use with boosting since they usually cause over fitting of the data [5]. One of the nice properties of AdaBoost.M1 algorithm is that it is less likely to encounter over fitting problems, since only a portion of the instance space is learned at each iteration using weak learners. In addition, ensemble of weak learners performs at least as well as a strong learner, but in considerably less time, since strong learners spend most of the training time during fine-tuning at lower error rates.

2.2 Connection to Incremental Learning

As shown in [4], the original distribution $\mathcal{D}$ of the data from which the training data S is drawn need not be known. Only a sample training dataset S drawn from $\mathcal{D}$ is provided to the learner. Consider a learner that has been trained with an initial dataset that came from a distribution $\mathcal{D}_1 \subseteq \mathcal{D}$ for which an initial set of hypotheses is generated. Now suppose that we are given a new dataset, drawn from a distribution $\mathcal{D}_2$, also unknown to the learner. If the original training dataset was a good representative of the entire instance space $\mathcal{D}$, from which all instances (training and testing) were drawn, then the classifier will perform well on the new dataset. If the classifier does not perform well on the new dataset, we can conclude that the first dataset was not a good representative of the entire instance space, and $\mathcal{D}_2 \not\subset \mathcal{D}_1$. In other words, the classifier was not trained with that part of the instance space from which the new data were drawn. Therefore, we can interpret these instances as *hard examples* of $\mathcal{D}$, and force the learner to learn instances coming from $\mathcal{D}_2$ by generating new hypotheses for this dataset and combine all hypotheses generated (for $\mathcal{D}_1$ and $\mathcal{D}_2$) for classification of unknown instances. The underlying assumption is that the unknown distribution $\mathcal{D}$ includes $\mathcal{D}_2$ as well as $\mathcal{D}_1$. Note that instances coming from $\mathcal{D}_1$ are learned through an initial set of hypotheses, and those coming from $\mathcal{D}_2$ are learned through a second set of hypotheses. All hypotheses are then combined by weighted majority voting.

Essentially, we use a modified version of AdaBoost.M1 to learn (possibly overlapping) different parts of the instance space using different training sets. Each new database represents a different region of the instance space, and each hypothesis learns that region of the space. If enough number of these hypotheses are generated, a weighted majority of these hypotheses can be used to determine the true concept (final hypothesis) being learnt.

2.3 An Incremental Learning Algorithm: LEARN++

Figure 1 illustrates the algorithm LEARN++. The arrow (←) indications point at major modifications made to AdaBoost.M1 given in [4].

Since AdaBoost.M1 targets improving classification accuracy, it is run on different distributions of a single training dataset, and the error is computed on the misclassified instances of the training set. Furthermore, distribution update is based on individual hypotheses h_t and their scaled errors β_t. LEARN++, however, runs a modified version of AdaBoost.M1 for each new dataset that become available. For each iteration t during current dataset $\mathcal{D}_k$, training (TR_t) and testing (TE_t) subsets are randomly generated from the current dataset.

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

- Sequence of m examples $S=[(x_1,y_1),(x_2,y_2),\ldots,(x_m,y_m)]$.
- Weak learning algorithm **WeakLearn** (MLP).
- Integer T_k, specifying the number of iterations.

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

Initialize $D_1(i) = 1/m$, $\forall i$.

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

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 a hypothesis $h_t : X \rightarrow Y$, and calculate the error of h_t : $\varepsilon_t = \sum_{i:h_t(x_i)\neq y_i} D_t(i)$ 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:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} B_t, & \text{if } H_t(x_i) = y_i \\ 1, & \text{otherwise} \end{cases}$$ ●

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++

These subsets are provided to **WeakLearn** (in this case, a weak MLP), which returns the hypothesis h_t. Unlike AdaBoost.M1, the error, ε_t, is computed from the misclassified patterns of $TR_t + TE_t$. If $\varepsilon_t > 1/2$, h_t is discarded, and new TR_t and TE_t are generated. The weighted majority voting is then called to compute the overall hypothesis, H_t, of all hypotheses generated. Also unlike AdaBoost.M1, the overall hypothesis, H_t, and its scaled error, B_t, are then used to update the distribution. Note that the weight of voting for each h_t is still based on its own scaled error β_t. These modifications were based on experimental observations made during simulations of the algorithm in an incremental learning setting.

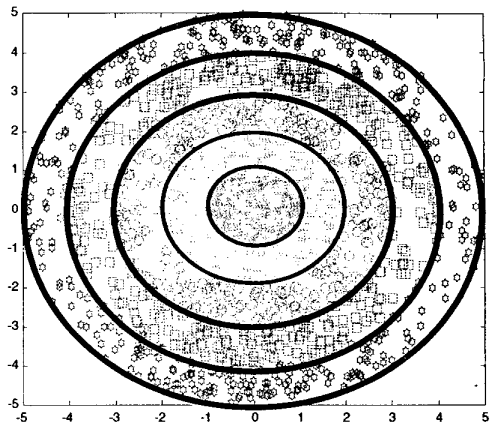

Figure 2. Circular regions dataset

3. SIMULATION RESULTS

LEARN++ was tested on a variety of synthetic and real world datasets. Due to space limitations, two representative ones are presented here.

3.1 Circular Regions Synthetic Dataset

Figure 2 illustrates the two-dimensional, five-class circular regions dataset, which consists of concentric circles. Innermost circle was labeled as class 1 and outermost circle was labeled as class five. Six training datasets, $S_1 \sim S_6$, were generated from this dataset, where S_1 and S_2 had instances from classes 1, 3, and 5, S_3 and S_4 added instances from class 4, and finally S_5 and S_6 added instances from class 2. An additional dataset, TEST, was also generated from all classes. These datasets were presented to the algorithm starting with S_1, and the performance of the overall hypothesis generated at the end of each training session (consisting of T iterations) was tested on the training dataset as well on the TEST dataset which was never seen by the algorithm during training. Note that once training with a particular dataset was completed, instances from that dataset were not shown to the algorithm during subsequent training sessions to assure a truly incremental learning setting.

Table 1 presents results obtained using this dataset. Each column represents the performance of final hypothesis obtained by computing the weighted majority of all hypotheses generated up to that point in all current and previous training sessions (TS). Each row shows the performance on a particular dataset. During training, TS1 used only S_1, TS2 used only S_2, etc. As expected, the performances of the hypotheses on their own training data was very high, but the performance on the TEST dataset was not satisfactory until later stages of incremental learning. Recall that TEST set which included instances from all classes was never seen during any training session. Also note that after TS3, when instances from an additional class (class 4) were introduced, the performance had a sudden jump from 59.8% to 72.2%. TS4 did not improve performance much (since no new classes were introduced in S_4), but TS5 did improve the performance, since instances from the last class (class 2) were included in S_5.

Although previous data are not used in subsequent training, previously generated hypotheses are used in the majority voting. However, using hypotheses only from last session performed around 70%, demonstrating the need for all previously generated hypotheses, hence the truly incremental nature of LEARN++.

Table 1. Performance of LEARN++ on synthetic data

Set	TS1	TS2	TS3	TS4	TS 5	TS6
S_1	99.7%	99.2%	98.4%	97.4%	96.7%	92.8%
S_2	-	96.1%	95.1%	93.8%	92.6%	88.5%
S_3	-	-	98.3%	98.3%	94.5%	93.5%
S_4	-	-	-	93.6%	92.5%	91.1%
S_5	-	-	-	-	84.0%	86.8%
S_6	-	-	-	-	-	88.8%
TEST	59.4%	59.8%	72.2%	73.4%	80.8%	88.0%

3.2 Gas Sensing Database

Quartz crystal microbalances (QCMs) are piezoelectric devices whose resonant frequencies change in response to a mass deposited on their surface. When QCMs are exposed to VOCs, the VOC molecules are deposited on the QCM surface causing a change in the resonant frequency of the crystal. An array of QCMs, each coated with a different polymer of varying affinities to various VOCs, are used to detect and identify VOCs from their frequency responses. The gas sensing dataset used in this study consisted of responses of six QCMs to five VOCs, including ethanol (ET), xylene (XL), octane (OC), toluene (TL), and trichloroethelene (TCE). The database consisted of 384 six-dimensional signals, half of which were used for training. This database was divided into three training datasets $S_1 \sim S_3$ and one test dataset, TEST. S_1 had instances from ET, OC and TL, S_2 had instances mainly from TCE (and very few from the previous three), and S_3 had instances from XL (and very few from the previous four). TEST set included instances from all classes. Table 2 presents the results, which is formatted similar to that of Table 1. Further details on VOC recognition using QCMs can be found in [6]. More information on this database, and sample signals are provided at the web site [7].

Table 2. Performance of LEARN++ on gas sensing data

Set	TS1	TS2	TS3
S_1	96.2%	77.5%	76.3%
S_2	-	87.5%	82.5%
S_3	-	-	90.0%
TEST	60.78%	70.1%	88.2%

As expected, the performances of the hypotheses on their own training datasets were high, whereas the performances on the TEST set improved incrementally as data with new classes were introduced in subsequent training sessions.

These results demonstrate that LEARN++ successfully converts MLP into an incremental learning algorithm, which otherwise suffers greatly from catastrophic forgetting.

4. SUMMARY AND DISCUSSIONS

We have introduced an incremental learning algorithm for MLP networks, which employs an ensemble of networks for learning new data. Initial results using this algorithm look very promising, but there is much room for improvement. Although LEARN++ was implemented using MLPs as weak learners, the algorithm itself is not dependent on the MLP, and current work includes evaluating LEARN++ using other learning algorithms.

Note that the algorithm has two key components. The first one is the selection of the subsequent training dataset (the distribution update rule). AdaBoost.M1 depends solely on the performance of individual h_t for distribution update, whereas LEARN++ uses the performance of overall H_t. The former one guarantees robustness and prevents performance deterioration, whereas the latter one allows efficient incremental learning capability when new classes are available. An appropriate combination of these might provide optimum performance levels.

The second key component is how individual hypotheses are combined, or how much each hypothesis should be weighted. LEARN++ uses weighted majority voting, however, various other schemes for combining hypotheses can be used. One such scheme could be learning the weight of each hypothesis through a subsequent learner, rather than estimating it from the scaled errors of individual hypotheses. Work is currently underway to address these issues.

Finally, the weighted majority voting for combining the hypotheses hints a simple way of estimating the reliability of the final decision and confidence limits of the performance figures. In particular, if a vast (marginal) majority of h_t agree on the class of a particular instance, then this can be interpreted as the algorithm having high (low) confidence in the final decision.

5. REFERENCES

[1] P. Jantke, "Types of incremental learning", *AAAI Symposium on Training Issues in Incremental Learning*, March 23-25, 1993, Stanford, CA.

[2] R. Schapire, "The strength of weak learning", *Machine Learing*, vol. 5, pp. 197-227, 1990.

[3] N. Littlestone, M.Warmuth, "The weighted majority algortithm", *Info. and Comp.*, vol. 108, pp. 212-261, 1994.

[4] Y. Freund, R. Schapire, "A decision theoretic generalization of online learning and application to boosting", *J. Comp. and Sys. Sci.*, vol.55, pp.119-139, 1997.

[5] T.G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, Boosting, and randomization, *Machine Learning*, (in press) 1999.

[6] E. Kress-Rogers, *Handbook of Biosensors and Electronic Nose. Medicine, Food and the Environment.* CRC Press: Boca Raton, FL., 1997.

[7] www.public.iastate.edu/~rpolikar/RESEARCH/vocdata.html