BOOSTING GAUSSIAN MIXTURES IN AN LVCSR SYSTEM

Geoffrey Zweig and Mukund Padmanabhan

IBM T. J. Watson Research Center
{gzweig,mukund}@watson.ibm.com

ABSTRACT

In this paper, we apply boosting to the problem of frame-level phone classification, and use the resulting system to perform voicemail transcription. We develop parallel, hierarchical, and restricted versions of the classic AdaBoost algorithm, which enable the technique to be used in large-scale speech recognition tasks with hundreds of thousands of Gaussians and tens of millions of training frames. We report small but consistent improvements in both frame recognition accuracy and word error rate.

1. INTRODUCTION

Boosting is a technique for sequentially training and combining a collection of classifiers in such a way that the later classifiers make up for the deficiencies of the earlier ones. Many variants exist [1, 7, 2, 3], but all follow the same basic strategy. There is a sequence of iterations, and at each iteration a new classifier is trained on a weighted set of the training examples. Initially, every example gets the same weight, but in subsequent iterations, the weights of hard-to-classify examples are increased relative to the easy ones. The outputs of the classifiers are then combined in such a way as to guarantee certain bounds on both training and testing error [2, 6]. Boosting algorithms have been successfully applied to a wide variety of problems, including a recent application to boosting neural nets in a continuous speech recognition system [8].

One of the main advantages of boosting is that it is possible to automatically generate very long streams of classifiers - anywhere from tens to thousands - that usually produce better and better composite performance. Although boosting was originally presented as a method for combining relatively weak classifiers, in this paper we focus on using the technique to combine state-of-the-art Gaussian-mixture systems. Unfortunately, these systems are very large, and generating and combining long streams of them requires some algorithmic modifications. The main contribution of this work is to develop and test parallel, hierarchical, and restricted variants of boosting that are suitable for use in extremely large systems.

This work was supported by DARPA Grant MDA972-97-C-0012.

2. THE ADABOOST ALGORITHM

In this section, we review the AdaBoost algorithm, and describe hierarchical and restricted extensions that allow for large speedups in training time. We base this work on the specific variant AdaBoost.M2 of [1], which we will refer to simply as AdaBoost.

2.1. Parallel AdaBoost for ML Classifiers

The input to the AdaBoost algorithm is a set of labeled training pairs, (x_i, y_i), where x_i represents the features associated with the ith example, and y_i is its label. In our application, the x_i are acoustic feature-vectors and the y_i are context-dependent phone labels. At each iteration t, a function $h_t(x, y)$ is learned that maps a feature/label pair into a number between 0 and 1. The function need not represent a probability distribution, though in our implementation it does. A weight, β_t, is assigned to each classifier, and the output of the composite classifier is given by

$$H(x,y) = \sum_t \left(\log \frac{1}{\beta_t}\right) h_t(x,y).$$

In our implementation, the atomic classifiers are mixtures of Gaussians, with one mixture for each context dependent phone. Denoting the jth Gaussian of phone i's mixture by G_i^j, and its mixture weight by m_i^j, and the prior on phone i by $P(i)$, the classifier learned at each iteration has the form:

$$h_t(x,y) = \frac{P(y)\sum_j m_y^j G_y^j(x)}{\sum_c P(c)\sum_j m_c^j G_c^j(x)}. \tag{1}$$

The AdaBoost algorithm maintains a distribution

```
For each iteration t:
  1. For each class c in parallel, weight and train:
     • $D_t(i,y) = \begin{cases} \frac{1}{k-1},\ y \neq c \text{ else } 0 & t = 0 \\ \frac{D_{t-1}}{Z_{t-1}} * \beta_{t-1}^{(1+h_{t-1}(x_i,c)-h_{t-1}(x_i,y))/2} & t > 0 \end{cases}$
     • Weight example $j$ as $w_t(j) = \sum_y D_t(j,y)$.
     • Train a new ML model for class $c$ with the weighted examples.
     • Store $Z_t^c = \sum_i \sum_y D_t(i,y)$.
  2. Compute $Z_t = \sum_c Z_t^c$
  3. For each class c in parallel, compute pseudoloss:
     $\epsilon_t^c = \frac{1}{2} \sum_i \sum_y \frac{D_t(i,y)}{Z_t}(1 - h_t(x_i,c) + h_t(x_i,y))$
  4. Compute total pseudoloss $\epsilon_t = \sum_c \epsilon_t^c$ and beta:
     $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$. If $\beta_t > 1$, terminate and use only
     iterations $1 \ldots t-1$.
```

Figure 1: Parallelized AdaBoost. In the parallelized sections, the index i refers only to examples of class c. k is the number of classes.

$D(i,y)$ over all possible example/label pairs (x_i, y) [1], and the goal at each iteration t is to produce a classifier h_t that minimizes the "pseudoloss":

$$\epsilon_t = \frac{1}{2} \sum_i \sum_y D_t(i,y)(1 - h_t(x_i, y_i) + h_t(x_i, y))$$

Minimizing this quantity has been shown to minimize upper bounds on training, and, in the binary-case, on testing error-rates [2, 6]. In order to train with a maximum-likelihood procedure, we replace the detailed distribution $D(i,y)$ with a simpler one that assigns a single weight to each example x_i, by summing over all class-wise weights: $w_t(i) = \sum_y D_t(i,y)$. Whereas minimizing the true pseudoloss would require a gradient descent technique - because changing the parameters associated with one class will affect all the others (see Equation 1) - this simplification allows us to train models for each class independently and in parallel, and is also used in [8]. The parallelized algorithm is presented in Figure 1. The quantity Z_t is a normalizing constant.

The exact form of the parallel algorithm is motivated by the fact that with thousands of classes and tens of millions of examples, it is impossible to store $D_t(i,y)$, and therefore each time it is used, it must be recomputed from scratch. In Figure 1, it is computed twice for each example - once in step 1, and once in step 3. In the original algorithm [1] (which is expressed in terms of five steps), it is used three times - once in step 1, once in step 3, and once in step 5. In the parallel algorithm, the distribution $D_t(i,y)$ is not actually normalized until step 3, but this delayed normalization does not affect the answer, since it is only the relative weights of the examples that matter during training.

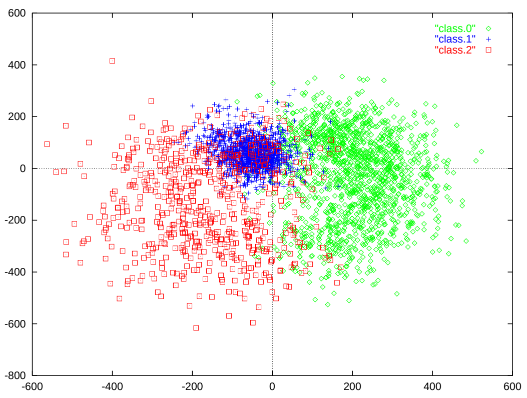

Figure 2: Training points.

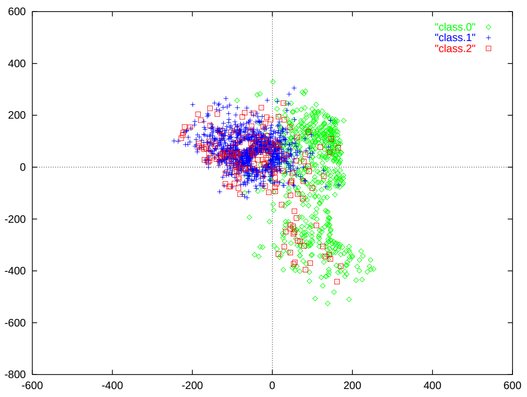

Figure 3: The points accounting for 90% of Z_t^c, after 20 iterations.

By setting h to a constant value, the pseudoloss will trivially achieve a value of 0.5, and beta a value of 1. Hence a functional classifier will always have a beta less than 1, ensuring that examples with large values for $h(x_i, y_i)$ will be de-emphasized in step 1.

Intuitively, the boosting algorithm works by placing high weight on hard-to-classify examples, and thereby focusing attention on the decision boundaries between classes. Figure 2 displays the points in a three class, two-dimensional classification task. Figure 3 shows the points accounting for 90% of the total weight of each class (i.e. 90% of Z_t^c) after 20 iterations.

The computational demands of AdaBoost are severe: in order to compute the pseudoloss or update

[1] $D(x_i, y_i)$ is always 0, so this is equivalent to a distribution over mislabel pairs.

```
Input: Phone confusion matrix $C$ in which $C(i,j)$ represents the fraction of occurrences of phone $i$ that are classified as phone $j$.
Initialize: Construct a set $\mathcal{S}$ of sets $S_i$ in which $S_i$ contains a single member representing phone $i$.
Define:
$W(i,j) = \sum_{Members(S_i)} \sum_{Members(S_j)} C(i,j), \ \forall i,j$
Repeat until $|\mathcal{S}| = k$: Merge the pair of sets $S_i$, $S_j$ for which $\frac{W(i,j)}{|S_i||S_j|}$ is maximum - remove $S_i$ and $S_j$ from $\mathcal{S}$ and replace them with $S_i \cup S_j$.
Output: $\mathcal{S}$.
```

Figure 4: Phone clustering algorithm. The output is a set of k phone clusters.

the weight distributions, it is necessary to compute $h_t(x_i, y)$ for all pairs of examples and classes. If there are f frames, g Gaussians learned per iteration, and i iterations, the runtime is $O(fgi)$. When the $D_t(i,y)$ are recomputed from scratch to conserve memory and disk space (thus requiring the computation of $h_0 \dots h_{t-1}$ in addition to h_t), the runtime is $O(fgi^2)$. This has motivated us to explore two streamlined variants of AdaBoost.

2.2. Restricted AdaBoost

The simplest way of reducing the computational load is to identify for each example x_i a small subset of "candidate" classes, and to assume that the h values for all the other classes will always be 0. We do this by computing $h_0(x_i, y)$ after the initial iteration for all classes y, and then defining the set of candidate classes as those that come within some fixed fraction of the maximum h value for x_i. In our implementation, we fixed the candidate-inclusion fraction at one part in a thousand, and roughly 50 out of the 2000 context-dependent phones ended up as candidates for each frame. Assuming that there are c candidates on average per frame, the runtime is reduced from $O(fgi^2)$ to $O(fg + fci^2)$.

2.3. Hierarchical AdaBoost

In the hierarchical version of AdaBoost, we partition the phones into clusters, and use boosting only to differentiate between the phones within a single cluster. Assuming there are m clusters of equal size, both in terms of the number of constituent phones and the number of training frames, the total computational load is reduced from $O(fgi^2)$ to $O(\frac{fgi^2}{m})$.

In the following, we will index clusters by k, and use $h^k(x,y)$ to refer to the level of plausibility assigned to the example-label pair (x,y) according to the models associated with cluster k. Further, k_y represents the cluster containing class y. In the hierarchical version of boosting, we compute $h(x_i, y)$ as

$$\begin{aligned} h(x_i,y) &= P(k_y|x_i)h^{k_y}(x_i,y) && (2)\\ &= \frac{P(k_y)P(x_i|k_y)}{\sum_f P(k_f)P(x_i|k_f)} h^{k_y}(x_i,y) && (3) \end{aligned}$$

The cluster priors $P(k_y)$ are determined as the fraction of the total training data belonging to any phone in cluster k_y. Using z to represent phones, $P(x|k_y)$ is computed as:

$$P(x|k_y) = \sum_{z \in k_y} P(z|k_y)P(x|z).$$

The quantity $P(z|k_y)$ is the fraction of training data within cluster k_y that belongs to phone z, and $P(x|z)$ is computed by evaluating the mixture of Gaussians associated with z. We use only the Gaussian mixtures from the first iteration of boosting to determine $P(x|k_y)$. This is because the Gaussians in subsequent iterations are placed near the decision boundaries of the phones within a cluster, and may not give a good representation of the points within the cluster as a whole.

To obtain the phone clusters, we use the bottom-up clustering scheme illustrated in Figure 4, and in our experiments used 15 clusters.

3. EXPERIMENTAL RESULTS

We evaluated our algorithms by testing them on a voicemail transcription task [5]. The training data consists of ordinary voicemail messages gathered at IBM, and the test set is 86 similar messages. Altogether, there are several million training frames, and 240 thousand test frames. The 39-dimensional acoustic vectors consisted of cepstra, deltas, and double-deltas. We used a baseline system with 134k Gaussians and 2313 context-sensitive phonetic units to do a Viterbi alignment of the data in order to get the examples for each phone that we then trained on. The baseline system is about as large a system as we can train with the available data.

After boosting, we used the composite H values to rank-order the phone-labels for each frame in the test data. These ranks were then fed into the standard IBM decoder [4]. In order to downweight silence and mumbling, we found it beneficial to equalize the phone priors in Equation 1 by initially weighting the examples of each class inversely proportional to the total number of examples in that class.

System	1st It.	2nd	3rd	4th	5th
69k-H	40.64%	40.29	39.77	-	-
69k-R	40.64	40.58	39.77	-	-
134k-H	39.71	39.73	39.74	-	-
134k-R	39.71	39.48	39.15	39.10	38.92

Table 1: Word error rates for hierarchical (H) and restricted (R) boosting.

System	1st It.	2nd	3rd	4th	5th
69k-H	16.3%	16.8	17.1	-	-
69k-R	16.3	16.8	17.0	-	-
134k-H	16.6	17.6	17.9		
134k-R	16.6	17.7	17.9	17.9	18.1

Table 2: Frame recognition rates.

Table 1 presents word-error results for two different sized hierarchical and restricted systems: one which we built from scratch by adding roughly 69k Gaussians at each iteration, and one in which we started with the 134k baseline system and added roughly 116k Gaussians per iteration. The smaller systems were run for 3 iterations for testing purposes. The larger hierarchical system was run for 3 iterations because one of the clusters contained a large number of classes with many associated frames; this was a bottleneck that prevented more iterations. Except for the larger hierarchical system, the word-error rates continually decrease. We hypothesize that the hierarchical scheme is more susceptible to overtraining because for a given class belonging to a given cluster, there are many examples that lie in regions of space that are associated with classes which are not cluster members. In the hierarchical scheme, these examples will receive low weight because it looks like they have no competition from other classes. In contrast, the restricted scheme is sensitive to all possible classes. To verify this, we looked at the candidate lists generated for each frame, and found that on average only 38% of the candidate classes are in the cluster associated with the frame's class. We believe that this is because with noisy data it is hard to find clusters that do not have a significant number of exceptions.

Table 2 shows the frame recognition rates on the alphabet of 2313 context-dependent phones, with the examples from each class weighted equally.

To get a notion of how sharply boosting focused on probable classes, at each iteration we computed the entropy H of the weight distribution over the examples for each phone. The quantity 2^H gives a measure of the effective number of examples for that phone at that iteration, and in Table 3, we show how the total number

System	1st It.	2nd	3rd	4th	5th
69k-H	1.0	0.996	0.989	-	-
69k-R	1.0	0.999	0.997	-	-
134k-H	1.0	0.995	0.986		
134k-R	1.0	0.999	0.996	0.992	0.988

Table 3: Normalized number of training examples.

of effective examples varied, normalized to a fraction of the total number of training frames. The fact that the hierarchical scheme focuses on a smaller number of examples is consistent with its observed tendency to overtrain.

4. CONCLUSION

The combination of theoretical expectations with experimental verification leads us to believe that boosting is an effective way of building large systems. Experimentally, we have found that hierarchical and restricted variants of the basic AdaBoost algorithm allow us to improve on very large systems.

5. REFERENCES

[1] Yoav Freund and Robert Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, Bari, Italy, July 1996. Morgan Kaufmann.

[2] Yoav Freund and Robert Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–137, 1997.

[3] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. Technical report, www-stat.stanford.edu/ hastie/Papers/, 1998.

[4] P.S. Gopalakrishnan, L.R. Bahl, and L.R. Mercer. A tree search strategy for large-vocabulary continuous speech recognition. In *ICASSP-95*, 1995.

[5] M. Padmanabhan, G. Saon, S. Basu, J. Huang, and G. Zweig. Recent improvements in voicemail transcription. In *Eurospeech-99*, 1999.

[6] R. Schapire, Y. Freund, P. Bartlett, and W.S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26(5):1651–1686, 1998.

[7] R. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. In *Proc. 11th Annual Conference on Computational Learning Theory*, 1998.

[8] Holger Schwenk. Using boosting to improve a hybrid hmm/neural network speech recognizer. In *ICASSP-99*, 1999.