ONLINE DISCRIMINATIVE LEARNING OF PHONEME RECOGNITION VIA COLLECTIONS OF GENERALIZED LINEAR MODELS

Koby Crammer

Dept. of Electrical Engineering,
The Technion - Israel Institute of Technology
Haifa 32000, Israel
koby@ee.technion.ac.il

Daniel D. Lee

Dept. of Electrical and Systems Engineering
University of Pennsylvania
Philadelphia, PA 19104, USA
ddlee@seas.upenn.edu

ABSTRACT

We describe a new online discriminative learning algorithm that efficiently and effectively recognizes phonemes in a speech sequence. The method builds upon recent work in online learning of a collection of generalized linear models using second order statistics of the model weight vectors. Evaluation on the TIMIT database shows that the algorithm achieves state-of-the-art phoneme recognition error rates compared to many other generative and discriminative models with the same expressive power.

1. INTRODUCTION

Automatic speech recognition (ASR) systems output a phonetic transcription of a given acoustic input utterance. For these systems, there is a large body of work using generalized linear models, either explicitly or implicitly (e.g. HMMs). Online learning algorithms are simple, fast, and require less memory compared to batch learning algorithms. Recent work has shown that they can also perform nearly as well as batch algorithms, making them quite attractive for large scale learning problems. In this work, we propose to use a collection of generalized linear models and train them simultaneously in an online manner. We represent this collection using the mean and covariance matrix of the underlying weight vectors and propose novel filtering-like learning algorithms with efficient regularization.

Learning models and techniques are commonly divided into two generic categories: generative and discriminative. For more than a decade most ASR systems were based on the framework of Hidden Markov Models (HMMs), which models the joint distribution of the acoustic signal and phonemes by factorizing it as a prior over the phonemes $\vec{p}$ and a conditional distribution of the acoustic signal $\vec{\boldsymbol{a}}$ given the underlying phonemes, $\Pr[\vec{p}, \vec{\boldsymbol{a}}] = \Pr[\vec{p}] \Pr[\vec{\boldsymbol{a}}|\vec{p}]$. Learning parameters for these models is then typically performed by maximizing the likelihood of the data.

Recently researchers have proposed using discriminative methods for building ASR systems, either by adapting the generative framework to optimize an error rate or related quantity (e.g. [1]), or by methods that replace the joint model with a conditional model of the phonemes *given* the acoustic signal [2, 3] . We focus here on a fully discriminative approach using a general linear model.

Recent work [4] in online learning showed how to guide learning via parametric information about the distribution of possible weight vectors. These algorithms are derived from minimizing the Kullback-Leibler divergence between Gaussian distributed weight vectors, and differ in the way they incorporate the training data. Empirical evaluation has demonstrated the advantages of this approach for a number of natural language processing problems [4].

We develop, analyze and evaluate a new learning algorithm for ASR that models the motions of individual weight vectors across a distribution of weight vectors. Similar to other online algorithms, our algorithm considers one acoustic utterance at a time, yet maintains a collection of weights rather than a single one. . Learning is defined as optimizing an appropriate velocity field among the weight vectors given the input utterance, such that the collective performance is improved while maintaining some invariant properties of the overall collection. Finding such a regularized velocity field results in a more robust online learning algorithm. We report experimental results showing that our approach yields phoneme recognition error rates that compare favorably with previous approaches.

2. METHODS

2.1. Problem Setting

In order to predict the sequence of phonemes, the acoustic input signal $\vec{\boldsymbol{a}}$ is represented as a sequence of feature vectors $\vec{\boldsymbol{a}} = (\boldsymbol{a}_1 \ldots \boldsymbol{a}_T)$, where T denotes the length of the utterance, and the features are points in a vector space $\boldsymbol{a}_t \in \mathbb{R}^d$. Each utterance is an acoustic realization of a sequence of phonemes, $\vec{p} = (p_1 \ldots p_\tau)$, where each phoneme belongs to a set of possible phonemes $p \in P$, $|P|$. Each phoneme in $\vec{p}$ may correspond to a different number of consecutive acoustic realizations in $\vec{\boldsymbol{a}}$. There exists a sequence of time-indices $t_1^s \ldots t_\tau^s$ (the starting time of each distinct phoneme) and a sequence $t_1^e \ldots t_\tau^e$ (the ending time of each distinct phoneme) such that $t_1^s = 1$ and $t_\tau^e = T$, and $t_k^s = t_{k-1}^e + 1$ (the acoustic representation of a phoneme starts immediately after the end of the acoustic representation of the previous phoneme). For simplicity, we write the phonetic vector explicitly as $\vec{p} = (p_1 \ldots p_T)$, where p_t is the phoneme corresponding to the acoustic signal $\boldsymbol{a}_t$, and which may contain consecutive sub-sequences with the same phoneme, i.e. $p_t = p$ for some $p \in P$ and $t = t_k^s \ldots t_k^e$.

A learning algorithm builds a mapping from signals $\vec{\boldsymbol{a}}$ (for all lengths T) to the corresponding phonemes $\vec{p}$, such that $\vec{p}$ captures the spoken utterance. Formally, $\hat{\vec{p}}$ denotes a prediction made by the algorithm for some input $\vec{\boldsymbol{a}}$. We assume the existence of a *loss* function that takes a phoneme prediction and a correct phoneme sequence (possibly of different lengths) and outputs a non-negative scalar representative of the difference between the two, i.e. , $\ell : P^* \times P^* \to \mathbb{R}^+$ where $P^* = \cup_{t=1}^T P^t$ (all finite-phonemic sequences).

2.2. Model

We build on the structured prediction approach [3] and use models that score pairs of input acoustic signals $\vec{\boldsymbol{a}}$ and candidate phoneme

sequences $\vec{p}$. Linear models for scoring are parameterized by a vector $\boldsymbol{w}$ and defined such that $s(\vec{\boldsymbol{a}}, \vec{p}; \boldsymbol{w}) = \boldsymbol{w} \cdot \phi(\vec{\boldsymbol{a}}, \vec{p})$ where $\phi(\vec{\boldsymbol{a}}, \vec{p}) \in \mathbb{R}^D$ is a feature vector representation of the acoustic-utterance and phonemic-sequence pair, and $\boldsymbol{w} \in \mathbb{R}^D$ is a vector of feature weights. Given the weight vector $\boldsymbol{w}$ and an acoustic input $\vec{\boldsymbol{a}}$ the algorithm outputs the phoneme sequence with the highest score,

$$f(\vec{\boldsymbol{a}}; \boldsymbol{w}) = \arg\max_{\vec{p}} s(\vec{\boldsymbol{a}}, \vec{p}; \boldsymbol{w}) = \arg\max_{\vec{p}} (\boldsymbol{w} \cdot \phi(\vec{\boldsymbol{a}}, \vec{p})) \ , \quad (1)$$

where the search is over all possible sequences.

The number of possible sequences is exponential with the length of the sequence T so even if the number of possible phonemes is not very large, naive search becomes intractable. Therefore, we constrain the feature function $\phi(\vec{\boldsymbol{a}}, \vec{p})$ to decompose over neighboring time units, so that it depends explicitly upon only two consecutive phonemes. The feature function is of the form, $\phi(\vec{\boldsymbol{a}}, \vec{p}) = \sum_{t=1}^{T-1} \phi(\vec{\boldsymbol{a}}, p_t, p_{t+1})$. This decomposition of the feature function can be related to the score function: $s(\vec{\boldsymbol{a}}, \vec{p}; \boldsymbol{w}) = \boldsymbol{w} \cdot \left\{ \sum_{t=1}^{T-1} \phi(\vec{\boldsymbol{a}}, p_t, p_{t+1}) \right\} = \sum_{t=1}^{T-1} (\boldsymbol{w} \cdot \phi(\vec{\boldsymbol{a}}, p_t, p_{t+1}))$. The Viterbi algorithm can then be used for efficient inference in time linear in T and quadratic in the number of possible phonemes $|P|^2$.

The framework of online confidence weighted (CW) learning was recently introduced [4] for binary classification. We follow CW modeling and represent the collection of weight vectors using the first two moments of the distribution: mean vector $\boldsymbol{w} \in \mathbb{R}^D$ and covariance matrix $\Sigma \in \mathbb{R}^{D \times D}$. Prediction is performed by substituting the value of the mean vector $\boldsymbol{w}$ and the given acoustic input $\vec{\boldsymbol{a}}$ in (1). Information in the covariance matrix is then used to guide the learning process, as described in the following section.

2.3. Learning

These algorithms are designed to work in online manner, learning in rounds (also called iterations). On the i-*th* round the algorithm receives an utterance represented with acoustic properties $\vec{\boldsymbol{a}}_i$ and outputs the best phoneme sequence $\hat{\vec{p}}_i$ according to its current model using the prediction rule of (1). The algorithm then receives the correct phoneme sequence $\vec{p}_i$ and its performance is evaluated using a local loss function $\ell\left(\vec{p}_i, \hat{\vec{p}}_i\right)$. For ASR, two loss functions are common: the Levenshtein edit distance between the correct sequence and the predicted one, called phoneme error (where $p_{i,t}$ is the phoneme in location t for sequence i), or the number of erroneous frames, $|\{t \ : \ \hat{p}_{i,t} \neq p_{i,t}\}|$. The algorithm then uses the last input pair, $(\vec{\boldsymbol{a}}_i, \vec{p}_i)$ to update the model. The goal of the algorithm is to minimize the cumulative loss, $\sum_t \ell\left(\vec{p}_i, \hat{\vec{p}}_i\right)$.

The only missing component in the description of the algorithm is the update rule: given the ith input pair $(\vec{\boldsymbol{a}}_i, \vec{p}_i)$, how to modify the current model $\boldsymbol{w}_i$ and Σ_i to get the next one $\boldsymbol{w}_{i+1}$ and Σ_{i+1}. We denote by $\mathbf{W}_i$ the random variable of weight vectors at round i, that is we have $\mathrm{E}[\mathbf{W}_i] = \boldsymbol{w}_i$ and $\mathrm{Cov}[\mathbf{W}_i] = \Sigma_i$. Our algorithms update the collection using a single linear transformation. Formally, $\mathbf{W}_{i+1} = A_i \mathbf{W}_i + \boldsymbol{b}_i$, where the algorithm selects the transformation A_i and $\boldsymbol{b}_i$. The matrix $A_i \in \mathbb{R}^{D \times D}$ represents stretching and rotating the distribution, and the vector $\boldsymbol{b}_i \in \mathbb{R}^D$ is an overall translation. For Gaussian distributions a linear transformation keeps the overall distribution Gaussian [5]. We set the parameters of the linear transformation A_i and $\boldsymbol{b}_i$ to minimize the following,

$$(A_i, \boldsymbol{b}_i) = \arg\min_{A, \boldsymbol{b}} \mathrm{E}_{\mathbf{W}_i} \left[\mathcal{C}_i (A\mathbf{W}_i + \boldsymbol{b}) \right] \text{ where } \quad (2)$$

$$\mathcal{C}_i(\mathbf{W}) = \frac{1}{2} (\mathbf{W} - \mathbf{W}_i)^\top \Sigma_i^{-1} (\mathbf{W} - \mathbf{W}_i) + C \left(\mathcal{L}_i(\mathbf{W}, \boldsymbol{\delta}_i) \right)^2 .$$

The sequence hinge-loss is given by

$$\mathcal{L}_i(\mathbf{W}, \boldsymbol{\delta}_i) = \max \left\{ 0, \ell\left(\vec{p}_i, \hat{\vec{p}}_i\right) - \mathbf{W} \cdot \boldsymbol{\delta}_i \right\} \quad (3)$$

$$\text{where } \boldsymbol{\delta}_i = \phi(\vec{\boldsymbol{a}}_i, \vec{p}_i) - \phi\left(\vec{\boldsymbol{a}}_i, \hat{\vec{p}}_i\right) , \quad (4)$$

which is the difference between the correct phoneme sequence feature vector and the predicted phoneme sequence feature vector.

Thus, $\mathcal{L}_i$ is a function of the mean parameters $\boldsymbol{w}$ and the vector $\boldsymbol{\delta}_i$. It equals the large-margin hinge loss suffered when comparing the score of the correct phoneme sequence and the predicted phoneme sequence. If $\boldsymbol{w}_i \cdot \phi(\vec{\boldsymbol{a}}_i, \vec{p}_i) < \boldsymbol{w}_i \cdot \phi\left(\vec{\boldsymbol{a}}_i, \hat{\vec{p}}_i\right) + \ell\left(\vec{p}_i, \hat{\vec{p}}_i\right)$ then $\mathcal{L}_i$ is proportional to the difference $\ell\left(\vec{p}_i, \hat{\vec{p}}_i\right) + \boldsymbol{w}_i \cdot \phi(\vec{\boldsymbol{a}}_i, \vec{p}_i) - \boldsymbol{w}_i \cdot \phi\left(\vec{\boldsymbol{a}}_i, \hat{\vec{p}}_i\right)$; and otherwise $\mathcal{L}_i = 0$.

The constant $C > 0$ is a parameter which controls the balance between two objectives. The first objective is for the weight vectors to be as close as possible to the current set, since the current set contains information retained from previous training examples. However, we also want the new set of parameters to do well on the current example. In particular, we use the predicted sequence of phonemes $\hat{\vec{p}}_i = \arg\max_{\vec{p}} \boldsymbol{w}_i \cdot \phi(\vec{\boldsymbol{a}}_i, \vec{p}_i)$ and modify the parameters $\boldsymbol{w}_i$ only if the score of the correct phoneme sequence $\vec{p}_i$ is not greater than the score of the predicted sequence $\hat{\vec{p}}_i$ by a difference of at least $\ell\left(\vec{p}_i, \hat{\vec{p}}_i\right)$, that is if $\boldsymbol{w}_i \cdot \phi(\vec{\boldsymbol{a}}_i, \vec{p}_i) < \boldsymbol{w}_i \cdot \phi\left(\vec{\boldsymbol{a}}_i, \hat{\vec{p}}_i\right) + \ell\left(\vec{p}_i, \hat{\vec{p}}_i\right)$ .In other words, we not only want the score of the correct phoneme sequence to be the highest, but we also want the difference between the score of the correct phoneme sequence and the second best phoneme sequence to be proportional to the loss suffered when predicting the latter instead of the former.

We analytically compute the expectation of the first term of $\mathcal{C}_i(\mathbf{W})$ using the derivatives of the second term to bound the expectation of $\mathcal{C}_i(\mathbf{W})$ and write the following bound on the objective of (2) explicitly,

$$\frac{1}{2} (\boldsymbol{w} - \boldsymbol{w}_i)^\top \Sigma_i^{-1} (\boldsymbol{w} - \boldsymbol{w}_i) + C \cdot \left(\mathcal{L}_i(\boldsymbol{w}, \boldsymbol{\delta}_i) \right)^2 + \quad (5)$$

$$\frac{1}{2} \mathrm{Tr}\left((A - I)^\top \Sigma_i^{-1} (A - I) \Sigma_i \right) + C \boldsymbol{\delta}_i^\top A \Sigma_i A^\top \boldsymbol{\delta}_i \ . \quad (6)$$

Note that the second term of (6) measures the variance in prediction after the update. Thus, the algorithm also implicitly reduces this uncertainty in the prediction. As we are interested in $\boldsymbol{w}_{i+1}$ and Σ_{i+1}, we perform a change of variables in the optimization. The updated variables are: A (as before) and $\boldsymbol{w}$, the mean of the modified collection. The relation between $\boldsymbol{b}$ and $\boldsymbol{w}$ is $\boldsymbol{w} = A\boldsymbol{w}_i + \boldsymbol{b}$.

The last optimization problem decomposes over $\boldsymbol{w}$ and A, the first line depends only on $\boldsymbol{w}$ while the second only on A. Clearly it is convex in both variables, and we compute the derivative of (5), set it to zero and solve for $\boldsymbol{w}$ to obtain,

$$\boldsymbol{w}_{i+1} \leftarrow \boldsymbol{w}_i + \frac{\mathcal{L}_i(\boldsymbol{w}_i, \boldsymbol{\delta}_i)}{\beta_i} \Sigma_i \boldsymbol{\delta}_i \ , \ \beta_i = \boldsymbol{\delta}_i^\top \Sigma_i \boldsymbol{\delta}_i + 1/C \ . \quad (7)$$

Before solving for A we note that the actual dimension D of all quantities involved is quite large, and thus it is not feasible to use a full matrix A (or Σ). Therefore, we employ only diagonal matrices. We now list three algorithmic variants of the update.

DIAG: Solving (6) exactly for diagonal matrices A and Σ_i we get an update rule for the rth diagonal element, $(A_i)_{r,r} = 1/\left(1 + C\delta_{i,r}^2 (\Sigma_i)_{r,r}\right)$, thus

$$(\Sigma_{i+1})_{r,r} = (\Sigma_i)_{r,r} / \left(1 + C\delta_{i,r}^2 (\Sigma_i)_{r,r}\right)^2 . \quad (8)$$

Input: training data $\{(\vec{\boldsymbol{a}}_i, \vec{p}_i)\}_{i=1}^m$ and parameter $C > 0$
Initialize: $\boldsymbol{w}_1 = \mathbf{0} \in \mathbb{R}^D$, $\Sigma_1 = I \in \mathbb{R}^{D\times D}$
For $i = 1, \ldots, m$

1. Compute best phoneme sequence $\hat{\vec{p}}_i = \arg\max_{\vec{p}} (\boldsymbol{w} \cdot \phi(\vec{\boldsymbol{a}}_i, \vec{p}_i))$
2. If $\ell\left(\vec{p}_i, \hat{\vec{p}}_i\right) > 0$ update (otherwise set $\boldsymbol{w}_{i+1} \leftarrow \boldsymbol{w}_i$ and $\Sigma_{i+1} \leftarrow \Sigma_i$) :
   - Update $\boldsymbol{w}_{i+1}$ using (7).
   - Equations used to compute β_i and update Σ_{i+1},

Method	Computing β_i	Updating Σ_{i+1}
DIAG	(7)	(8)
APPROX	(7)	(10)
MATCH	(11)	(10)

Output: Averaged weight vector $\frac{1}{m}\sum_i^m \boldsymbol{w}_{i+1}$; matrix Σ_{m+1}.

Fig. 1. DIAG, APPROX and MATCH updates for online ASR learning.

The denominator of (8) is always greater than zero and thus equation 8 is well-defined. Also note that the diagonal elements of Σ_i goes to zero with i. Intuitively, the more examples the algorithms processes, the lower its uncertainty about the correct value of $\boldsymbol{w}$ is.

APPROX: We solve (6) exactly for full matrices and then take only the diagonal part. As we shall see below, the performance of this approximated update is competitive with other algorithms. For this update, $A_i = I - \Sigma_i \boldsymbol{\delta}_i \boldsymbol{\delta}_i^\top C / \left(1 + C\boldsymbol{\delta}_i \Sigma_i \boldsymbol{\delta}_i^\top\right)$. Substituting the last equation in the inverse term $\Sigma_{i+1}^{-1} = \left(A\Sigma_i A^\top\right)^{-1}$ we get,

$$\Sigma_{i+1}^{-1} = \left(A\Sigma_i A^\top\right)^{-1} = \Sigma_i^{-1} + \left(2C + C^2 \boldsymbol{\delta}_i^\top \Sigma_i \boldsymbol{\delta}_i\right) \boldsymbol{\delta}_i \boldsymbol{\delta}_i^\top . \quad (9)$$

We use Woodbury's identity to compute the updated covariance,

$$\Sigma_{i+1} = \Sigma_i - \Sigma_i \boldsymbol{\delta}_i \boldsymbol{\delta}_i^\top \Sigma_i \frac{C^2 \boldsymbol{\delta}_i \Sigma_i \boldsymbol{\delta}_i^\top + 2C}{(1 + C\boldsymbol{\delta}_i^\top \Sigma_i \boldsymbol{\delta}_i)^2} . \quad (10)$$

Finally, we extract the diagonal part of the last matrix, which is equivalent to replacing $\Sigma_i \boldsymbol{\delta}_i \boldsymbol{\delta}_i^\top \Sigma_i$ with its diagonal part.

MATCH: The last two updates were based upon previous work [5] for binary prediction but applied here more generally to temporal sequences. The third update rule is motivated from a desire to provide a novel theoretical analysis for the algorithm, by replacing β_i from (7) with

$$\beta_i = \left((1 + C\boldsymbol{\delta}_i^\top \Sigma_i \boldsymbol{\delta}_i)^2\right) / \left(C^2 \boldsymbol{\delta}_i \Sigma_i \boldsymbol{\delta}_i^\top + 2C\right) , \quad (11)$$

which is the inverse coefficient of $\Sigma_i \boldsymbol{\delta}_i \boldsymbol{\delta}_i^\top \Sigma_i$ in (10) above.

Pseudocode of the algorithm appears in Fig. 1. The algorithm returns an *average* of all the weight vectors $\boldsymbol{w}_i$ computed during the runtime of the algorithm. Empirically, the average weight-vector performs better than a single weight vector (including the last one $\boldsymbol{w}_{m+1}$), and it can also be justified from a theoretically point of view.

3. ANALYSIS

As mentioned above, the MATCH update is derived by analyzing the mistake bound in the worst case setting. We start with the following lemma, that extends Lemma 3 of [4] both for sequential algorithms and variable learning rate. The proof is omitted due to lack of space.

Lemma 1 *Let $\chi_i = \boldsymbol{\delta}_i^\top \Sigma_i \boldsymbol{\delta}_i$, and $\gamma_i = 1/\left(2C + C^2\chi_i\right)$. Then, for every round i for which an update occurs, the following two equalities hold for all vectors $\boldsymbol{u} \in \mathbb{R}^D$,*

$$\boldsymbol{u}^\top \Sigma_i^{-1} \boldsymbol{w}_i = \boldsymbol{u}^\top \Sigma_{i-1}^{-1} \boldsymbol{w}_{i-1} + \boldsymbol{u}^\top \boldsymbol{\delta}_i / \gamma_i$$
$$\boldsymbol{w}_i^\top \Sigma_i^{-1} \boldsymbol{w}_i = \boldsymbol{w}_{i-1}^\top \Sigma_{i-1}^{-1} \boldsymbol{w}_{i-1} + \left(\chi_i + \gamma_i - \mathcal{L}_i^2 \gamma_i\right) / \left(\gamma_i \left(\chi_i + \gamma_i\right)\right)$$

We now state the main theorem of the analysis.

Theorem 2 *Denote by $R = \sup_i \|\boldsymbol{\delta}_i\|$. For any weight $\boldsymbol{u} \in \mathbb{R}^d$, the cumulative loss suffered by the algorithm is upper bounded by*

$$\sum_i \ell\left(\vec{p}_i, \hat{\vec{p}}_i\right) \le \sqrt{\frac{2 + CR^2}{4C}} \sqrt{\boldsymbol{u}^\top \Sigma_{m+1}^{-1} \boldsymbol{u}} \sqrt{\log\left(\det\left(\Sigma_{m+1}^{-1}\right)\right)} + \left(1 + CR^2/2\right) \sum_i \mathcal{L}\left(\boldsymbol{u}, \boldsymbol{\delta}_i\right) .$$

In other words, the cumulative loss suffered by the algorithm is bounded by the cumulative (hinge) loss of any fixed-weight vector multiplied by a factor greater than one, plus an additional term, $\sqrt{\frac{2+CR^2}{4C}} \sqrt{\boldsymbol{u}^\top \Sigma_{m+1}^{-1} \boldsymbol{u}} \sqrt{\log\left(\det\left(\Sigma_{m+1}^{-1}\right)\right)}$ that is sub-linear in the number of utterances m, $\mathcal{O}\left(\|\boldsymbol{u}\| \sqrt{m \log m}\right)$. The proof is omitted due to lack of space. This theorem provides a first bound on the mistakes made by the MATCH algorithm, that is a variant of previous algorithms [5] for which analysis was not included.

4. EMPIRICAL EVALUATION

The performance of the above algorithms was evaluated using the TIMIT corpus. We used a standard partition [1] of the corpus into a training set of $3,696$ utterances, test set of 400 utterances, and validation set of 192 utterances. We mapped the 61 phonemes of TIMIT into a subset of size 48. We used the pre-processing procedure of [1] and generated representations with 39 Mel-frequency coefficients together with their first and second derivatives.

Setting: We used two types of feature mapping. In the first set of experiments we followed [6] and set $\phi(\vec{\boldsymbol{a}}, \vec{p})$ either to be $\phi_n(\vec{\boldsymbol{a}}_t, p)$ (node features) for all $t = 1 \ldots T$ and some phoneme p or $\phi_e(p_t, p_{t+1})$ (edge features) for all $t = 1 \ldots T - 1$. The first type captures the relation between a possible phoneme and the corresponding acoustic input (equivalent to the emission probabilities of HMMs). The second type captures the dependency between two adjacent phonemes, (and is analogous to the transition probabilities in HMMs. In particular, for the first type we used functions of the 39 acoustic features, and for the second type we used a single fixed feature. In the second set of experiments we used all the features from the previous experiment, and also added correlations between the 39 acoustic features at time t and the acoustic features at time $t-1$ and $t+1$, yielding an additional $2 \cdot 39^2 = 3,042$ features.

Two metrics were used for algorithm evaluation [1]: phoneme error rate, computed using the Levenshtein distance, and the frame error rate, which is the fraction of misclassified frames. The former better reflects the ultimate goal of speech recognition. We followed a common practice and mapped the 48 states into 39 categories.

Three variants of the algorithm were evaluated: DIAG, APPROX and MATCH. All algorithms were run and averaged for 20 epochs with 3 different values for the parameter C. For each algorithm we picked the best value of C and epoch using the validation set and report the results on the test set. We also quote the results of other systems and methods, including two HMM systems [3, 1] (with a single out state setting), a kernel-based recognizer trained with a PA algorithm [3] denoted by KSBSC, and a PA that uses simpler features [6]. Online [1] and batch LM-HMM [7] large-margin training of HMMs,

Method	Frame Error Rate	Phone Error Rate
HMM (1 mixture component)[1]	39.9	42.0
CSS (1 mixture component)[1]	**28.3**	33.5
PA[6]	30.0	33.4
DROP[6]	29.2	**31.1**
DIAG	32.9	43.7
APPROX	29.4	30.0
MATCH	**29.2**	**29.7**
HMM[3]	35.1	40.9
Online LM-HMM (8 mixtures)[1]	**25.0**	30.2
Batch LM-HMM (8 mixtures)[7]	-	**28.2**
KSBSC[3]	-	45.1
PAC-Bayes 1-frame[8]	27.7	30.2
CRFs (9-frames MLP)[9]	-	29.3
PAC-Bayes 9-frames[8]	**26.5**	**28.6**
DIAG 3-frames-context	32.3	45.4
APPROX 3-frames-context	**27.8**	**27.9**
MATCH 3-frames-context	28.0	28.0

Table 1. Frame error rates and phone error rates for few existing methods compared with our proposed three methods, grouped according to the amount of information present in the feature representations.

a recently proposed PAC-Bayes 1-frame [8], and DROP [6], which is most similar to our algorithm.

Results: Our main findings are summarized in Table 1. The first column shows the frame error rate and the second column the phone error rate. The first block of 4 rows summarizes previous results of algorithms that use a representation equivalent to HMM with a single mixture component. CD-HMMs trained with an online method [1] have the smallest frame error rate, and DROP has a significantly lower phoneme error rate compared to all other methods.

The next block of 3 rows summaries the performance of our three proposed algorithms. Surprisingly, the DIAG update performs worse, the APPROX update is better, and MATCH, the version with the theoretical guarantee, performs the best. Comparing to previous methods with the same expressive power (first block), we see that CD-HMMs has the lowest frame error rate, although MATCH (and DROP [6]) are almost comparable. Furthermore, the phone error rate of the MATCH algorithm (29.7) is the best, and even better than the performance of CD-HMMS with eight mixture components (30.2) [1], although it maintains and updates much less parameters.

The third block of 5 rows summaries the performance of other algorithms that use more complex features over a single frame. The HMM [3] uses left-to-right HMM of 5 emitting states with 40 diagonal Gaussians. Online and Batch LM-HMM use eight mixture components, and both KSBSC and PAC-Bayes 1-frame employ a kernel. Indeed, most of the methods reported in this block achieve lower frame error rate (Online HM-HMM) or lower phone error rate (Batch HM-HMM) than methods presented in the previous two blocks.

The $4th$ and $5th$ blocks show the results with methods that also use the acoustic features of *neighbor* frames. The $4th$ block quotes results from previous methods that use the features of 9 frames. The PAC-Bayes 9-frame achieves the best performance, which is only slightly worse than Batch LM-HMM with 8 mixtures. Finally, the performance of our algorithms using the second set of features with larger context (only 3 frames), are reported in the $5th$ block. Unlike results with a single frame, here APPROX outperforms MATCH. Evaluated using the more meaningful phone error rate, both MATCH and APPROX achieve the *best* performance compared to all algorithms.

One possible perspective that differentiates between methods appears in Fig. 2. The left panel shows the cumulative distribution of the number of phone-segments per utterance. The three lines correspond to the distributions according to the true test data prediction

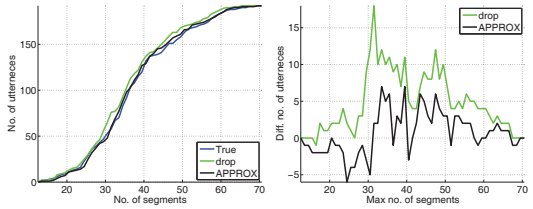

Fig. 2. Left: cumulative distribution of number of phone segments per utterance for the true distribution of data, and as predicted by DROP[6] and APPROX. Right: the difference between the distributions of DROP and the true labels, and APPROX and the true labels. See text.

using DROP [6] and APPROX. Each point shows the total number of utterances with segments less or equal to the corresponding value on the x-axis. For example, there are 53 utterances with 30 or less segments in the test data, yet there are such 65 utterances when predicting with DROP and 48 when predicting with APPROX. From the panel we observe that the line corresponding to DROP is always higher than the line of the true distribution, which indicates that the distribution of DROP is biased towards predictions with shorter segmentations. On the other hand, the line corresponding to APPROX better tracks the true distribution. These observations are emphasized in the right panel of Fig. 2 in which we plot the difference between each of the lines corresponding to DROP and APPROX and the true labels. That is, the $y = 0$ axis corresponds to the distribution of the test set. Clearly, the DROP methods are biased towards shorter predictions, whereas the APPROX method better tracks the true distribution, and the maximal difference between the two cumulative distributions is only 7 (as opposed to 18 for DROP).

These results indicate the possibility of training ASR models using simpler and more efficient online learning algorithms, without sacrificing performance. We are currently working to extend theoretical guarantees across the various algorithms, and to evaluate their performance on larger speech databases.

5. REFERENCES

[1] C.C. Cheng, F. Sha, and L. Saul, "A fast online algorithm for large margin training of continuous-density hidden markov models," in *INTERSPEECH*, 2009.

[2] A. Gunawardana, M. Mahajan, A. Acero, and J.C. Platt, "Hidden conditional random fields for phone classification," in *INTERSPEECH*, 2005.

[3] J. Keshet, S. Shalev-Shwartz, S. Bengio, Y. Singer, and D. Chazan, "Discriminative kernel-based phoneme sequence recognition," in *INTERSPEECH*, 2006.

[4] K. Crammer, A. Kulesza, and M. Dredze, "Adaptive regularization of weight vectors," in *NIPS 23*, 2009.

[5] K. Crammer and Daniel D. Lee, "Learning via gaussian herding," in *NIPS*, 2010.

[6] K. Crammer, "Efficient online learning with individual learning-rates for phoneme sequence reco.," in *ICASSP*, 2010.

[7] F. Sha and L. Saul, "Comparison of large margin training to other discriminative methods for phonetic recognition by hidden markov models," in *ICASSP*, 2007.

[8] J. Keshet, D. McAllester, and T. Hazan, "Pac-bayesian approach for minimization of phoneme error rate," in *ICASSP*, 2011.

[9] Jeremy Morris and Eric Fosler-Lussier, "Conditional random fields for integrating local discriminative classifiers," *IEEE Tran. On Audio Speech and Lang. Proc.*, vol. 16, 2008.