The subspace Gaussian mixture model – a structured model for speech recognition

Daniel Povey[a], Lukáš Burget[b], Mohit Agarwal[c], Pinar Akyazi[d], Feng Kai[e], Arnab Ghoshal[f], Ondřej Glembek[b], Nagendra Goel[g], Martin Karafiát[b], Ariya Rastrow[h], Richard C. Rose[i], Petr Schwarz[b], Samuel Thomas[h]

[a]*Microsoft Research, Redmond, WA, USA*
[b]*Brno University of Technology, Czech Republic*
[c]*IIIT Allahabad, India*
[d]*Boğaziçi University, Istanbul, Turkey*
[e]*Hong Kong University of Science and Technology, Hong Kong, China*
[f]*Saarland University, Saarbrücken, Germany*
[g]*Go-Vivace Inc., Virginia, USA*
[h]*Johns Hopkins University, Baltimore, MD, USA*
[i]*McGill University, Montreal, Canada*

Abstract

We describe a new approach to speech recognition, in which all Hidden Markov Model (HMM) states share the same Gaussian Mixture Model (GMM) structure with the same number of Gaussians in each state. The model is defined by vectors associated with each state with a dimension of, say, 50, together with a global mapping from this vector space to the space of parameters of the GMM. This model appears to give better results than a conventional model, and the extra structure offers many new opportunities for modeling innovations while maintaining compatibility with most standard techniques.

Keywords: Speech Recognition, Gaussian Mixture Model, Subspace

Email addresses: dpovey@microsoft.com (Daniel Povey), burget@fit.vutbr.cz (Lukáš Burget), mohit.iiita@gmail.com (Mohit Agarwal), pinarakyazi@hotmail.com (Pinar Akyazi), tony.fengkai@gmail.com (Feng Kai), aghoshal@lsv.uni-saarland.de (Arnab Ghoshal), glembek@fit.vutbr.cz (Ondřej Glembek), nagendra.goel@govivace.com (Nagendra Goel), karafiat@fit.vutbr.cz (Martin Karafiát), ariya@jhu.edu (Ariya Rastrow), rose@ece.mcgill.ca (Richard C. Rose), schwarzp@fit.vutbr.cz (Petr Schwarz), samuel@jhu.edu (Samuel Thomas)

1. Introduction

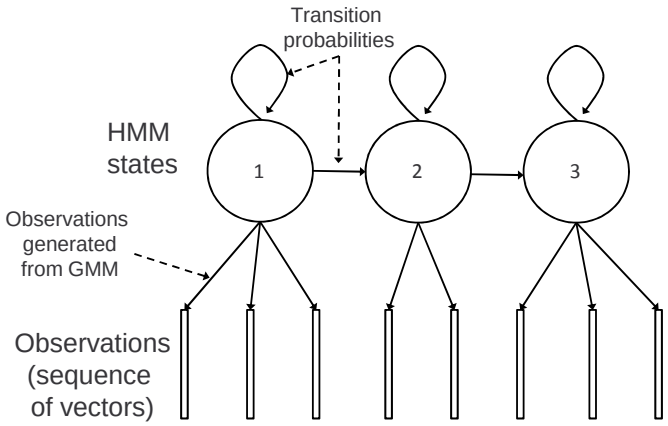
Speech recognition based on the Hidden Markov Model-Gaussian Mixture Model (HMM-GMM) framework generally involves training a completely separate GMM in each HMM state. We introduce a model in which the HMM states share a common structure but the means and mixture weights are allowed to vary in a subspace of the full parameter space, controlled by a global mapping from a vector space to the space of GMM parameters. We call this a Subspace GMM (SGMM). This method has similarities to Joint Factor Analysis as used in speaker recognition [19] and to Eigenvoices [21] and Cluster Adaptive Training (CAT) [10] as proposed for speech recognition.

In this paper we aim to present this technique in sufficient detail for the reader to be able to replicate our results and to understand the ideas used in deriving the formulae; however, we omit derivations. These can be found in the technical report [26]. In the experimental section, we will show that this modeling approach can outperform conventional models. It has a particular advantage where the amount of in-domain data available to train the model is small, because out-of-domain and even out-of-language speech data can be used to train the parameters of the global mapping.

1.1. HMM-GMM based speech recognition

In HMM-GMM based speech recognition (see [11] for review), we turn the short-time spectral characteristics of speech into a vector (the "observations" of Figure 1, sometimes called frames), and build a generative model that produces sequences of these vectors. A left-to-right three-state HMM topology as in Figure 1 will typically model the sequence of frames generated by a single phone. Models for sentences are constructed by concatenating HMMs for sequences of phones. Different HMMs are used for phones in different left and right phonetic contexts, using a tree-based clustering approach to model unseen contexts [40]. We will use the index j for the individual context-dependent phonetic states, with $1 \leq j \leq J$. While J could potentially equal three times the cube of the number of phones (assuming we model just the immediate left and right phonetic context), after tree-based clustering it will typically be several thousand. The distribution that generates a vector within

Figure 1: HMM for speech recognition



HMM state j is a Gaussian Mixture Model (GMM):

$$p(\mathbf{x}|j) = \sum_{i=1}^{M_j} w_{ji} \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_{ji}, \boldsymbol{\Sigma}_{ji}). \tag{1}$$

Table 1 shows the parameters of the probability density functions (pdfs) in an example system of this kind: each context-dependent state (of which we only show three rather than several thousand) has a different number of sub-states M_j.

Table 1: Parameters for pdfs in baseline HMM system

State 1	State 2	State 3
$\boldsymbol{\mu}_{11}, \boldsymbol{\Sigma}_{11}, w_{11}$ $\boldsymbol{\mu}_{12}, \boldsymbol{\Sigma}_{12}, w_{12}$ $\boldsymbol{\mu}_{13}, \boldsymbol{\Sigma}_{13}, w_{13}$	$\boldsymbol{\mu}_{21}, \boldsymbol{\Sigma}_{21}, w_{21}$ $\boldsymbol{\mu}_{22}, \boldsymbol{\Sigma}_{22}, w_{22}$ $\boldsymbol{\mu}_{23}, \boldsymbol{\Sigma}_{23}, w_{23}$ $\boldsymbol{\mu}_{24}, \boldsymbol{\Sigma}_{24}, w_{24}$	$\boldsymbol{\mu}_{31}, \boldsymbol{\Sigma}_{31}, w_{31}$ $\boldsymbol{\mu}_{32}, \boldsymbol{\Sigma}_{32}, w_{32}$

Table 2: Parameters for pdfs in basic SGMM system (derived quantities in gray)

	State 1	State 2	State 3
	$\mathbf{v}_1$	$\mathbf{v}_2$	$\mathbf{v}_3$
$\mathbf{M}_1, \mathbf{w}_1, \mathbf{\Sigma}_1$	$\boldsymbol{\mu}_{11}, \Sigma_{11}, w_{11}$	$\boldsymbol{\mu}_{12}, \Sigma_{12}, w_{12}$	$\boldsymbol{\mu}_{13}, \Sigma_{13}, w_{13}$
$\mathbf{M}_2, \mathbf{w}_2, \mathbf{\Sigma}_2$	$\boldsymbol{\mu}_{21}, \Sigma_{21}, w_{21}$	$\boldsymbol{\mu}_{22}, \Sigma_{22}, w_{22}$	$\boldsymbol{\mu}_{23}, \Sigma_{23}, w_{23}$
$\mathbf{M}_3, \mathbf{w}_3, \mathbf{\Sigma}_3$	$\boldsymbol{\mu}_{31}, \Sigma_{31}, w_{31}$	$\boldsymbol{\mu}_{32}, \Sigma_{32}, w_{32}$	$\boldsymbol{\mu}_{33}, \Sigma_{33}, w_{33}$
$\mathbf{M}_4, \mathbf{w}_4, \mathbf{\Sigma}_4$	$\boldsymbol{\mu}_{41}, \Sigma_{41}, w_{41}$	$\boldsymbol{\mu}_{42}, \Sigma_{42}, w_{42}$	$\boldsymbol{\mu}_{43}, \Sigma_{43}, w_{43}$

The SGMM also has a GMM within each context-dependent state, but instead of specifying the parameters directly we specify a vector $\mathbf{v}_j \in \mathbb{R}^S$ in each state together with a global mapping from this S-dimensional vector space to the pdf parameters. The simplest form of our model can be expressed in the following three equations:

$$p(\mathbf{x}|j) = \sum_{i=1}^{I} w_{ji} \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_{ji}, \mathbf{\Sigma}_i) \tag{2}$$

$$\boldsymbol{\mu}_{ji} = \mathbf{M}_i \mathbf{v}_j \tag{3}$$

$$w_{ji} = \frac{\exp \mathbf{w}_i^T \mathbf{v}_j}{\sum_{i'=1}^{I} \exp \mathbf{w}_{i'}{}^T \mathbf{v}_j} \tag{4}$$

where $\mathbf{x} \in \mathbb{R}^D$ is the feature vector and $j \in \{1 \dots J\}$ is the context-dependent speech state (hereafter, just "speech state"). The model for speech state j is a GMM with I Gaussians (typically $200 \leq I \leq 2000$), with covariance matrices $\mathbf{\Sigma}_i$ which are shared between states, mixture weights w_{ji} and means $\boldsymbol{\mu}_{ji}$. Table 2 shows the parameters for our SGMM system, with derived parameters in gray. The parameters $\boldsymbol{\mu}_{ij}, \mathbf{\Sigma}_{ij}, w_{ij}$ are derived from $\mathbf{v}_j$ together with $\mathbf{M}_i, \mathbf{\Sigma}_i, \mathbf{w}_i$ (this is a slight simplification; there is a normalization factor for the weights that breaks this model; also $\mathbf{\Sigma}_{ij} = \mathbf{\Sigma}_i$ so we do not use the notation $\mathbf{\Sigma}_{ij}$ elsewhere). We use the term "subspace" to indicate that the parameters of the GMMs are limited to a sub-space of the entire space of parameters of an I-dimensional mixture model[1]. Equation (4) for the weights

[1]Technically this is only true if we take the parameters to be the means and the unnormalized log weights

requires some justification. We note that the denominator of (4) is necessary for normalization; we also note that if we replaced exp with any other function (bar the pointless generalization $f(x) = k_1 \exp k_2 x$), the (negated) auxiliary function we construct during E-M would no longer be guaranteed convex and this would cause difficulties in optimization[2]. If we were to declare the individual weights w_{ji} to be parameters of the model instead of using this formula, the model size would be dominated by weights which we consider undesirable; also, a Maximum Likelihood estimation framework would no longer be sufficient: it would lead to zero weights, and in combination with a pruning scheme we describe below, this might lead to zero likelihoods.

Note that within this document, quantities that are denoted by the same letter but have different typeface or different numbers of subscripts or superscripts are distinct unless otherwise stated, so for instance $\mathbf{v}_{jm}$ and $\mathbf{v}^{(s)}$ are different quantities.

1.3. Context and prior work

Our modeling approach is inspired by prior work in speaker recognition. Text-independent speaker recognition systems often use a GMM, sometimes referred to as a "Universal Background Model" (UBM), which is adapted via Maximum a Posteriori (MAP) to individual speakers [32]. Our approach also features a UBM, and it has strong similarities to the Joint Factor Analysis (JFA) approach used for speaker recognition [19]. The work we describe here grew out of previous work applying speaker identification techniques to speech recognition. In that work, Maximum a Posteriori (MAP) adaptation was used to adapt the UBM to each speech state [30]. Although the Word Error Rate (WER) results were promising, the resulting systems had a very large number of parameters. We also briefly described an earlier version of this technique, in [29, 33]. The mathematical details for those papers were published by technical report [27]. We have also published the mathematical details corresponding to this current paper, in the technical report [26]. Some of the work described here has been published in conference papers [28, 12, 4, 14].

1.4. Model extensions

We described the most basic form of the SGMM in Equations (2) to (4). Because that model uses a very small number of parameters to describe

[2]Thanks to Patrick Nguyen.

each speech state (i.e. S parameters, where S is the subspace dimension, typically around 50), we introduce a mechanism to increase the state-specific parameters. We introduce "sub-states", where a state j will have M_j sub-states indexed $1 \leq m \leq M_j$, each with its own vector $\mathbf{v}_{jm}$ and associated sub-state weight $c_{jm} \geq 0$ with $\sum_{m=1}^{M_j} c_{jm} = 1$. The intuition is that the vectors $\mathbf{v}_{jm}$ should correspond to a particular point in phonetic space, and if a phoneme-in-context can be realized in multiple phonetically distinct ways then we should use a mixture of these vectors. Replacing the single index j with the pair j, m, the model is:

$$p(\mathbf{x}|j) = \sum_{m=1}^{M_j} c_{jm} \sum_{i=1}^{I} w_{jmi} \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_{jmi}, \boldsymbol{\Sigma}_i) \tag{5}$$

$$\boldsymbol{\mu}_{jmi} = \mathbf{M}_i \mathbf{v}_{jm} \tag{6}$$

$$w_{jmi} = \frac{\exp \mathbf{w}_i^T \mathbf{v}_{jm}}{\sum_{i'=1}^{I} \exp \mathbf{w}_{i'}^T \mathbf{v}_{jm}}. \tag{7}$$

Our distribution in each state is now a mixture of mixtures, with M_j times I Gaussians in state j. We now introduce further extensions relating to speaker adaptation. We use a Constrained MLLR (CMLLR)[3] feature transformation, as in [8], whereby we replace the feature $\mathbf{x}$ with the transformed feature

$$\mathbf{x}' = \mathbf{A}^{(s)} \mathbf{x} + \mathbf{b}^{(s)} \tag{8}$$

where s is the speaker index. The CMLLR transform also introduces a log determinant factor $|\det \mathbf{A}^{(s)}|$ into the likelihood. Constrained MLLR is a very standard technique and we are just introducing the appropriate notation to combine it with our method. At this stage we also describe a speaker-adaptation technique that is tied to our SGMM framework. We add a speaker-dependent offset in the mean for each Gaussian index i (see the term $\mathbf{N}_i \mathbf{v}^{(s)}$ below). This is equivalent to training a separate speaker-dependent offset on the features for each index i, but using the "subspace" idea to combine the entire set of offsets into a single low-dimensional per-

[3] Also known as fMLLR

speaker parameter $\mathbf{v}^{(s)}$. The equations become:

$$p(\mathbf{x}|j,s) = |\det \mathbf{A}^{(s)}| \sum_{m=1}^{M_j} c_{jm} \sum_{i=1}^{I} w_{jmi} \mathcal{N}(\mathbf{x}'; \boldsymbol{\mu}_{jmi}^{(s)}, \boldsymbol{\Sigma}_i) \quad (9)$$

$$\boldsymbol{\mu}_{jmi}^{(s)} = \mathbf{M}_i \mathbf{v}_{jm} + \mathbf{N}_i \mathbf{v}^{(s)} \quad (10)$$

$$w_{jmi} = \frac{\exp \mathbf{w}_i^T \mathbf{v}_{jm}}{\sum_{i'=1}^{I} \exp \mathbf{w}_{i'}^T \mathbf{v}_{jm}}. \quad (11)$$

The symmetry of (10) mirrors the two-factor approach of Joint Factor Analysis [19]; there are also similarities to Eigenvoices [21] and Cluster Adaptive Training [10]. The new speaker-specific parameter $\mathbf{v}^{(s)} \in \mathbb{R}^T$ is a vector of similar size to $\mathbf{v}_{jm}$, i.e. $T \simeq S$. This is a very simple form of adaptation: the contributions from the speaker and the speech state are simply added together. The weights are determined as before and are not affected by $\mathbf{v}^{(s)}$; we have not yet experimented with speaker-dependent mixture weights[4]. In test time the vectors $\mathbf{v}^{(s)}$ would typically be learned by Maximum Likelihood using a speaker independent decoding as supervision. In training time $\mathbf{N}_i$ and $\mathbf{v}^{(s)}$ would be learned by E-M along with all the other parameters; this is the same idea as Speaker Adaptive Training [3].

In this paper we will also make use of an extension to the CMLLR framework, in which the CMLLR matrix $\mathbf{W}^{(s)} = \left[\mathbf{A}^{(s)}; \mathbf{b}^{(s)}\right]$ is represented as a sum of basis matrices:

$$\mathbf{W}^{(s)} = \mathbf{W}_0 + \sum_{b=1}^{B} a_b^{(s)} \mathbf{W}_b, \quad (12)$$

where $\mathbf{W}_b$ are the basis matrices and $\mathbf{W}_0$ just represents the "default" matrix $[\mathbf{I}; \mathbf{0}]$. This is the same model as [38], but we use a novel optimization method to train the CMLLR matrix which naturally leads to an easy and efficient way to estimate and use the basis matrices $\mathbf{W}_b$, removing some of the practical difficulties of the scheme described in [38]. This is useful if we want to do CMLLR adaptation where the amounts of data to be adapted on are small. Note that the basis representation of (12) along with our novel optimization

[4]Speaker-dependent mixture weights are challenging to use efficiently; however, it seems to be possible to solve this problem by structuring the likelihood evaluation in the right way. We may experiment with this in future.

method may also be employed with a conventionally structured system; we will describe this in a separate paper.

1.5. Universal Background Model (UBM)

We mentioned that in speaker identification there is the notion of a Universal Background Model (UBM). This may be thought of as a generic mixture of Gaussians that models all classes. It is typically adapted by Maximum A Posteriori (MAP) to the class (i.e., speaker) [32]. The UBM does not appear in our equations above but it is still needed for initialization of our system (it would not be practical to train a model like this from scratch). We also use the UBM to prune the Gaussian indices i on each frame; that is, we evaluate the UBM likelihoods and retain, say, the top 10 scoring indices. We note that the UBM and the number of Gaussians I are fixed throughout training of the SGMM itself; the UBM training is a separate initialization process.

1.6. Typical parameter count

Table 3: Example system sizes: $J = 1500$, $D = 39$

(a) Baseline: $M_j = 18$

Parameter Name	Parameter Count	Example Count
State-specific parameters		
$\boldsymbol{\mu}_{jm}$	$DJ\times$ #mix-comp	1 053 000
$\boldsymbol{\Sigma}_{jm}$ (diag)	$DJ\times$ #mix-comp	1 053 000
c_{jm}	$J\times$ #mix-comp	27 000
Total		2 133 000

(b) SGMM: $S = 40$, $M_j = 5$

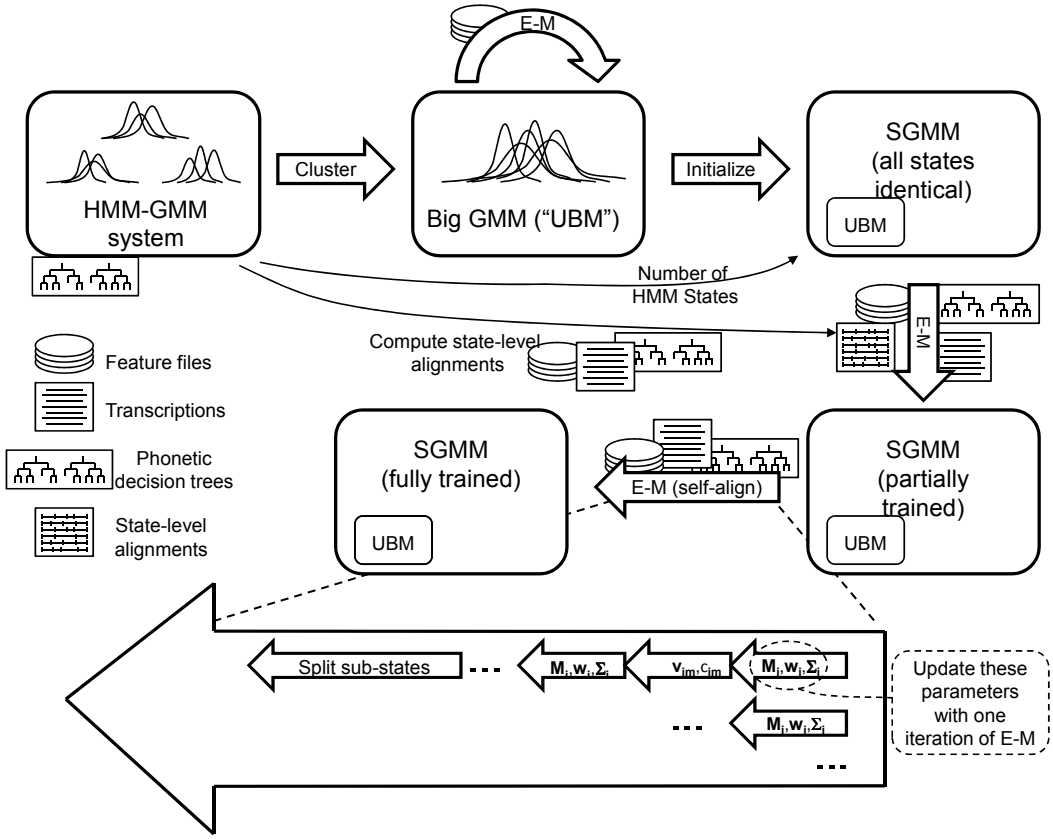
Parameter Name	Parameter Count	Example Count
Global parameters		
$\mathbf{M}_i$	IDS	624 000
$\boldsymbol{\Sigma}_i$	$ID(D+1)/2$	312 000
$\mathbf{w}_i$	IS	16 000
State-specific parameters		
$\mathbf{v}_{jm}$	$S\sum_j M_j$	300 000
c_{jm}	$\sum_j M_j$	7500
Total (no UBM)		1 259 500
UBM parameters		
$\boldsymbol{\Sigma}_i$	$ID(D+1)/2$	312 000
$\bar{\boldsymbol{\mu}}_i$	ID	15 600
$\bar{w}_i$	I	400
Total (w/ UBM)		1 587 500

We illustrate the ideas with a concrete example at this point, so the reader can get a sense of the dimensions of the quantities involved. Tables 3(a) and 3(b) show the sizes of a reasonably well optimized conventional and SGMM-based system respectively, trained on about 10 hours of data (this corresponds roughly to our English CallHome system). The SGMM system has sub-states but no speaker adaptive parameters. The parameter count of the SGMM system, excluding the UBM, is about 1.25 million, versus about 2.1 million in the baseline, but the amount of parameters that are tied to specific speech states is much smaller in the SGMM system at only about 0.38 million, which is a factor of five fewer than the baseline. Because of this it is possible to use out-of-domain data to train the shared parameters, and to use a smaller quantity of more relevant data to train the state-specific parameters. Also, because there are strongly diminishing returns to increasing I or S, when we have more training data a well tuned SGMM system will tend to be much more compact than a well tuned conventional system.

1.7. Overview of training procedure

We now briefly summarize the training procedure we used; we will provide further details in Section 6. Figure 2 shows the process. We start with a traditional HMM-GMM system. This is how we obtain our phonetic context tying information (the decision trees), and by clustering the Gaussians in this system we have a convenient way to initialize the UBM. Experiments done with the MAP-based setup described in [30] showed that this kind of initialization seemed to improve Word Error Rate (WER) versus training the UBM from scratch. The HMM-GMM system is also used to obtain alignments for early iterations of SGMM training. This was simply the most convenient setup for us; there is no essential dependency on a traditional system. The first steps are to initialize the UBM via a clustering process and to refine it using E-M (without class labels) on the training data. After that we initialize the SGMM system; we do this in such a way that all the states' pdfs are equivalent to the UBM. Next there are two overall phases of E-M training: the first uses the (Viterbi) state alignments of our baseline HMM-GMM, and the second uses Viterbi alignments obtained from the SGMM itself. Each phase is iterative. The diagram expands the second phase of SGMM training (self aligned) to show the iterations. We alternate training different parameter types on different iterations, as in Speaker Adaptive Training (SAT) [3]. The diagram is a slight idealization; in practice we are more aggressive and update more parameter types than the theory allows (see Section 6). Period-

Figure 2: SGMM training procedure used here



ically we split the sub-states $\mathbf{v}_{jm}$; this is analogous to the Gaussian splitting procedure in normal E-M training [39].

1.8. Expectation-Maximization for the SGMM

Our training is based on Expectation-Maximization. On each iteration we accumulate statistics from all our training data and then maximize an auxiliary function. It is not possible to compactly store sufficient statistics that would allow us to train all parameter types on the same iteration and guarantee an increase in likelihood. Particular pairs, such as $\mathbf{M}_i$ and $\mathbf{v}_{jm}$, are problematic here. Instead we alternate the update of different types of parameter on different iterations.

The E-M updates for some parameter types, for example the matrices $\mathbf{M}_i$, are quite straightforward and are equivalent to maximizing a quadratic auxiliary function. Something that does require some care here is that the (negated) auxiliary function is not always positive definite. Our approach is to leave the parameters the same in the null-space of the quadratic part of the auxiliary function. Identifying null-spaces of matrices is hard in the presence of roundoff errors [15], so we use a method that does this in a soft manner without explicitly identifying the null-space (Appendix A).

The E-M updates for those parameters $\mathbf{v}_{jm}$ and $\mathbf{w}_i$ that determine the weights w_{jmi} are a little more complicated. The problem is the normalizing (denominator) term of (7). Our solution to this problem involves a combination of auxiliary function inequalities and a quadratic approximation to an auxiliary function.

Our E-M update for the CMLLR matrix, used for speaker adaptation, is not the same as the traditional approach used for diagonal models [8]. The problem is that we use full, not diagonal, covariance matrices. Although techniques have been described that handle this case [35, 31], they are not very efficient. Our approach is more efficient and also makes it easy to represent the CMLLR matrix as a sum of basis matrices as in (12).

1.9. Strengths and weaknesses of the SGMM

One of the strengths of the SGMM is that it is relatively compact; in particular, the number of parameters that are associated with specific speech states is quite small. This enables training with less data, and makes it possible to take advantage of out-of-domain or even out-of-language data to train the shared parameters. In experiments previously carried out at IBM (see forthcoming book chapter [29]), this type of model gave more improvements with a small amount of training data (50 hours) than with 1000 hours. However, it still outperformed a conventional model with 1000 hours of data.

There appears to be a qualitative difference between the SGMM and conventional models: the optimal language-model scaling factor is closer to unity than with a normal model (e.g. 10 rather than 13, in the experiments described here). We choose to interpret this as arising from more accurate, less noisy likelihoods.

The chief weakness of this type of model is that it is substantially more complex to implement than the conventional one. In many situations, something as complex as this would have to be very substantially better than the baseline to make it worth implementing. Although we show improvements

here, the improvements after Speaker Adaptive Training are probably not yet of the required magnitude. One of the objectives of this paper is to make it as easy as possible for others to implement these methods.

The flip side of this complexity is that the model has extra structure that makes further developments easier. For instance, the shared index i makes it possible to implement the speaker adaptation framework in which we add a term $\mathbf{N}_i\mathbf{v}^{(s)}$ to the mean. There are many further extensions that one can consider; for example, introducing a two-level structure (with two indices replacing i) with different amounts of sharing at different levels. Our hope is that with further work, both modeling improvements and tuning, the advantages of this type of system will become substantial enough to make the extra work worthwhile.

1.10. Overview of the rest of this document

From this point we describe the SGMM framework in more detail, including all the key equations necessary to implement it. Due to space concerns we generally omit derivations (see [26]), but we attempt to explain the basic principles behind the derivations where they are not obvious. The order of sections is dictated to some extent by the desire to always refer to earlier rather than later equation numbers; this is why we put fast likelihood evaluation (Section 3) before the section on accumulation (Section 4).

Section 2 describes how we initialize the UBM and the SGMM. Section 3 describes our methods for fast likelihood evaluation in the SGMM framework. Section 4 provides accumulation equations for all parameter types (the Expectation step of an E-M procedure). Section 5 describes the Maximization steps for all parameter types. Section 6 summarizes the overall E-M procedure and provides details on our typical training schedule. Section 7 explains our data-sets and experimental setup. Section 8 contains experimental results, and we conclude in Section 9. In Appendix A, we describe some optimization procedures for quadratic auxiliary functions. Appendix Appendix B provides the details of the Constrained MLLR update.

2. Initialization

The first stage of initialization is to obtain a trained UBM. Next we obtain from the UBM a feature-normalizing transform that is used during SGMM initialization; next, we initialize the SGMM itself.

2.1. UBM initialization and training

The Universal Background Model (UBM) is a mixture of Gaussians with means $\bar{\boldsymbol{\mu}}_i$, full covariances $\bar{\boldsymbol{\Sigma}}_i$ and weights $\bar{w}_i$. In our experiments, we first initialized this by clustering the diagonal Gaussians in the baseline HMM-GMM system to obtain a mixture of diagonal Gaussians. The clustering algorithm we use differs from the one described in [26], for efficiency reasons. Initially we create a large GMM consisting of all the (diagonal) Gaussians in the baseline HMM set, with weights proportional to the weights within the HMM states multiplied by the state occupation probabilities. Then, while the number of Gaussians is more than the desired number, we merge the pair of Gaussians i and j that would result in the least log-likelihood reduction, computed as follows where k is the index of the merged Gaussian:

$$\Delta\mathcal{L} = \tfrac{w_i}{2}\log\det\boldsymbol{\Sigma}_i + \tfrac{w_j}{2}\log\det\boldsymbol{\Sigma}_j - \tfrac{w_k}{2}\log\det\boldsymbol{\Sigma}_k \quad (13)$$

$$w_k = w_i + w_j \quad (14)$$

$$\boldsymbol{\mu}_k = (w_i\boldsymbol{\mu}_i + w_j\boldsymbol{\mu}_j)/w_k \quad (15)$$

$$\boldsymbol{\Sigma}_k = \mathrm{diag}\left(\tfrac{w_i}{w_k}(\boldsymbol{\Sigma}_i + \boldsymbol{\mu}_i\boldsymbol{\mu}_i^T) + \tfrac{w_j}{w_k}(\boldsymbol{\Sigma}_j + \boldsymbol{\mu}_j\boldsymbol{\mu}_j^T) - \boldsymbol{\mu}_k\boldsymbol{\mu}_k^T\right) \quad (16)$$

Since evaluating the cost of merging each pair of Gaussians in the original HMM set would be too expensive, during an initial stage we only allow the merging of Gaussians in "similar" states, with similarity being judged by the metric of Equation (13) applied to Gaussians obtained by merging all the Gaussians in each state (we assign equal weights to all these merged Gaussians when evaluating (13) for this purpose).

After clustering, we train the resulting GMM with 8 E-M iterations of full-covariance re-estimation on all of the speech data. In each update we set the weights to all be the same, to encourage even distribution of data among the Gaussians. We limit the condition number of variance matrices to 10^5 by flooring eigenvalues, and remove Gaussians if too many of their eigenvalues are floored in this way (our limit was 5); in practice this only results in the removal of one or two Gaussians.

2.2. Feature normalizing transform

A feature normalizing transform $\mathbf{J} \in \mathbb{R}^{D\times D}$ (this is the same as $\mathbf{T}^{-1}$ in [26]) which is similar to the inverse of an LDA transformation but without dimensionality reduction, is computed prior to initializing the model. This transform is needed during model initialization to enable us to handle the

case where $S < D+1$ or $T < D$ in a non-arbitrary way, and also later on if we want to increase the phonetic or speaker subspace dimensions S or T. We compute:

$$\begin{aligned}
\boldsymbol{\Sigma}_W &= \sum_{i=1}^{I} \bar{w}_i \bar{\boldsymbol{\Sigma}}_i && (17)\\
\boldsymbol{\mu} &= \sum_{i=1}^{I} \bar{w}_i \bar{\boldsymbol{\mu}}_i && (18)\\
\boldsymbol{\Sigma}_B &= \left(\sum_{i=1}^{I} \bar{w}_i \bar{\boldsymbol{\mu}}_i \bar{\boldsymbol{\mu}}_i^T \right) - \boldsymbol{\mu}\boldsymbol{\mu}^T && (19)\\
\boldsymbol{\Sigma}_W &= \mathbf{L}\mathbf{L}^T \text{ (Cholesky decomposition)} && (20)\\
\mathbf{S} &= \mathbf{L}^{-1}\boldsymbol{\Sigma}_B\mathbf{L}^{-T} && (21)\\
\mathbf{S} &= \mathbf{U}\mathbf{D}\mathbf{V}^T \text{ (SVD)} && (22)\\
\mathbf{J} &= \mathbf{L}\mathbf{U} && (23)
\end{aligned}$$

We require that the Singular Value Decomposition be sorted by decreasing singular value. From this procedure we only need to retain $\mathbf{J}$.

2.3. Model initialization

The goal of model initialization is to provide a good starting point for E-M optimization. We initialize the model so that the GMM in each state j is identical to the starting UBM. We also ensure that the matrices $\mathbf{M}_i$ and $\mathbf{N}_i$ have reasonable values so that the training of the vectors $\mathbf{v}_{jm}$ can get started. We initialize with just one sub-state per state. When we initialize, the subspace dimensions S and T will be specified by the user. A typical configuration is to set $S = D+1$ and $T = 0$, since the speaker subspace can be initialized at a later iteration. We require that $S \leq D+1$ and $T \leq D$. The initialization is:

$$\begin{aligned}
M_j &= 1 && (24)\\
c_{j1} &= 1 && (25)\\
\mathbf{v}_{j1} &= \mathbf{e}_1 \in \mathbb{R}^S && (26)\\
\mathbf{M}_i &= [\ \bar{\boldsymbol{\mu}}_i\ \mathbf{j}_1 \ldots \mathbf{j}_{S-1}\ ] && (27)\\
\mathbf{N}_i &= [\ \mathbf{j}_1 \ldots \mathbf{j}_T\ ] && (28)\\
\mathbf{w}_i &= \mathbf{0} \in \mathbb{R}^S && (29)\\
\boldsymbol{\Sigma}_i &= \bar{\boldsymbol{\Sigma}}_i && (30)
\end{aligned}$$

where $\mathbf{e}_1 = [1\ 0\ \ldots\ 0]$ is a unit vector in the first dimension, and $\mathbf{j}_d$ is the d'th column of the matrix $\mathbf{J}$ computed in (23). This initialization is such that the initial values of $\boldsymbol{\mu}_{j1i}$ are the same as the UBM means $\bar{\boldsymbol{\mu}}_i$, and the elements of the state vectors $\mathbf{v}_{j1}$ and the speaker vectors $\mathbf{v}^{(s)}$ are interpreted as offsets on the means in the LDA-normalized space. The transform $\mathbf{J}$ ensures that when not using the full dimensionality of the feature space, we always take the most important directions.

3. Likelihood evaluation

3.1. Overview

Naïvely implemented, this style of model would be very inefficient to evaluate likelihoods with. It is not possible to store the expanded model's means in memory; computing the means for a state j on the fly and evaluating all the likelihoods in the most obvious way would take about $2M_jIDS + M_jID^2 = 10.8$ million flops in our example, versus only about $3M_jI = 2106$ flops for the baseline setup. However, it is possible to make the speeds comparable by a combination of clever computation and pruning. We prune the indices I using the UBM to the most likely 10 or 20 indices, and organize the computation in such a way that the extra computation in flops for each Gaussian evaluated is $2S$ where S is the subspace dimension. Pruning to 10 indices, this amounts to 5000 flops in our example system; however, being a simple dot product it will typically be faster per flop than the baseline. Below we describe how we organize the computation for fast likelihood evaluation.

3.2. Global and speaker-specific pre-computation

We need to compute a quantity n_{jmi} for each Gaussian in the system. This contains data-independent terms in the log-likelihood:

$$\begin{aligned} n_{jmi} &= \log c_{jm} + \log w_{jmi} \\ &\quad -\tfrac{1}{2}\left(\log\det\boldsymbol{\Sigma}_i + D\log(2\pi) + \boldsymbol{\mu}_{jmi}^T\boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_{jmi}\right) \end{aligned} \tag{31}$$

using $\boldsymbol{\mu}_{jmi} = \mathbf{M}_i\mathbf{v}_{jm}$. This requires a significant amount of computation and should be done during model update to avoid repeating the work each time a process reads in the model. The quantities n_{jmi} actually dominate the memory requirements of this model; in our example, the number of these quantities is $I\sum_j M_j = 3\,000\,000$, which is about twice as large as the model.

If using the speaker vectors $\mathbf{v}^{(s)}$, then for each speaker we also need to compute the offsets:

$$\mathbf{o}_i^{(s)} = \mathbf{N}_i \mathbf{v}^{(s)}. \tag{32}$$

3.3. Per-frame Gaussian selection and pre-computation

On each frame we first compute the adapted features, if we are doing CMLLR adaptation:

$$\mathbf{x}'(t) = \mathbf{A}^{(s)} \mathbf{x}(t) + \mathbf{b}^{(s)}. \tag{33}$$

Next we use the UBM to compute a set of P pruned indices $i_1 \dots i_P$. In order to avoid doing a full covariance computation on all of the I Gaussians in the UBM, we first use the diagonal versions of the covariances $\bar{\mathbf{\Sigma}}_i^{\text{diag}}$ which are the same as $\bar{\mathbf{\Sigma}}_i$ but with the off-diagonal elements set to zero. We compute the diagonal versions of the likelihoods, $\bar{w}_i \mathcal{N}(\mathbf{x}'(t) - \mathbf{o}_i^{(s)} | \bar{\boldsymbol{\mu}}_i, \bar{\mathbf{\Sigma}}_i^{\text{diag}})$, take the top P^{diag} indices, compute the selected full likelihoods $\bar{w}_i \mathcal{N}(\mathbf{x}'(t) - \mathbf{o}_i^{(s)} | \bar{\boldsymbol{\mu}}_i, \bar{\mathbf{\Sigma}}_i)$, and select the top P indices $\{i_1 \dots i_P\}$. We used $P^{\text{diag}} = 50$, $P = 15$.

For each $i \in \{i_1 \dots i_P\}$, we next compute and store:

$$\begin{aligned}
\mathbf{x}_i(t) &= \mathbf{x}'(t) - \mathbf{o}_i^{(s)} && (34)\\
\mathbf{z}_i(t) &= \mathbf{M}_i^T \mathbf{\Sigma}_i^{-1} \mathbf{x}_i(t) && (35)\\
n_i(t) &= \log |\det \mathbf{A}^{(s)}| - \tfrac{1}{2} \mathbf{x}_i(t)^T \mathbf{\Sigma}_i^{-1} \mathbf{x}_i(t) && (36)
\end{aligned}$$

where $\mathbf{x}_i(t)$ is the speaker-adapted version of the features used for Gaussian index i, the dot product of $\mathbf{z}_i(t)$ with $\mathbf{v}_{jm}$ gives the "cross-term" in the likelihood, and $n_i(t)$ contains likelihood terms for index i and frame t.

3.4. Gaussian likelihood computation

The log-likelihood for state j, sub-state m and Gaussian i is:

$$\log p(\mathbf{x}(t), m, i | j) = n_i(t) + n_{jmi} + \mathbf{z}_i(t) \cdot \mathbf{v}_{jm}. \tag{37}$$

The total log-likelihood for state j is:

$$\log p(\mathbf{x}(t) | j) = \log \sum_{m,i} p(\mathbf{x}(t), m, i | j). \tag{38}$$

These kinds of summations should be done using a "log add" function that computes $f(a, b) = \log(\exp a + \exp b)$ without ever calculating $\exp a$ or $\exp b$ directly.

4. Accumulation for SGMM training

4.1. Overview

In this section, we describe the various quantities that are accumulated from the data during SGMM training. This is the Expectation step of an E-M training procedure. We present this as if we were updating all parameter types on each iteration, but in fact from a theoretical point of view we cannot update all parameter types on each iteration. We do not discuss which types of parameters can be updated together since in practice it is usually better to combine them more aggressively than the E-M theory predicts is possible, so we are content to treat it an empirical issue. See [26, Section 6.3] for more discussion. In our implementation, the set of parameter types to be updated is defined by a set of flags supplied on each iteration of training; we will describe a typical setup in Section 6.

4.2. Posterior computation

Accumulation for all the different parameter types requires posteriors over the individual Gaussians. These can be computed from the per-Gaussian and per-state likelihoods of Equations (37) and (38) respectively:

$$\begin{aligned} \gamma_{jmi}(t) &\equiv p(j, m, i|t) \\ &= p(j|t)\frac{p(\mathbf{x}(t), m, i|j)}{p(\mathbf{x}(t)|j)} \end{aligned} \tag{39}$$

where $p(j|t) \equiv \gamma_j(t)$ will be supplied by some standard forward-backward or Viterbi algorithm. In our implementation, $\gamma_j(t)$ is a zero or one posterior based on a Viterbi alignment which on the first few iterations of training is obtained using likelihoods from the baseline GMM system, and thereafter is obtained using the SGMM itself.

In our implementation we did not do any further pruning on these posteriors computed in Equation (39) over and above pruning to the top P indices i, because we recompute the Viterbi alignment on each iteration and the training time is dominated by this. However, it is possible to speed up the actual accumulation by further pruning because most of the posteriors $\gamma_{jmi}(t)$ are extremely small. An effective randomized pruning algorithm that preserves the expected value of statistics is described in [26, Section 10.1]. When pruning the posteriors, care must be taken to replace quantities which represent a number of frames, e.g. appearing in Equation (51), with totals of pruned posteriors which may differ slightly from the frame counts.

4.3. Accumulation for model parameters

The count statistics γ_{jmi} should be computed on all iterations of E-M as they appear in the updates of most of the parameter types:

$$\gamma_{jmi} = \sum_t \gamma_{jmi}(t). \tag{40}$$

The accumulation needed to re-compute the vectors $\mathbf{v}_{jm}$ is:

$$\mathbf{y}_{jm} = \sum_{t,i} \gamma_{jmi}(t)\mathbf{z}_i(t) \tag{41}$$

with $\mathbf{z}_i(t)$ as given by Equation (35). This is the main linear term in the auxiliary function for $\mathbf{v}_{jm}$; the quadratic term can be worked out from the counts γ_{jmi} and the model itself.

The statistics for the model projections $\mathbf{M}_i$ are

$$\mathbf{Y}_i = \sum_{t,j,m} \gamma_{jmi}(t)\mathbf{x}_i(t){\mathbf{v}_{jm}}^T. \tag{42}$$

These are also used for re-estimation of the variances $\mathbf{\Sigma}_i$.

The statistics for the speaker projections $\mathbf{N}_i$ are analogous to the statistics needed to update the model projections $\mathbf{M}_i$. We first define a speaker-space analogue to $\mathbf{x}_i(t)$, in which we treat the main mean term as an offset to be subtracted from the features:

$$\mathbf{x}_{jmi}(t) = \mathbf{x}'(t) - \boldsymbol{\mu}_{jmi} \tag{43}$$

using $\boldsymbol{\mu}_{jmi} = \mathbf{M}_i\mathbf{v}_{jm}$. We accumulate:

$$\mathbf{Z}_i = \sum_{t,j,m} \gamma_{jmi}(t)\mathbf{x}_{jmi}(t){\mathbf{v}^{(s(t))}}^T \tag{44}$$

where $s(t)$ is the speaker active on frame t. We also have to accumulate some weighted outer products of the speaker vectors:

$$\mathbf{R}_i = \sum_{t,j,m} \gamma_{jmi}(t)\mathbf{v}^{(s(t))}{\mathbf{v}^{(s(t))}}^T \tag{45}$$

$$= \sum_s \left(\sum_{t \in \mathcal{T}(s),j,m} \gamma_{jmi}(t) \right) \mathbf{v}^{(s)}{\mathbf{v}^{(s)}}^T \tag{46}$$

where $s(t)$ is the speaker active on frame t, and we use $\mathcal{T}(s)$ to represent the set of frames valid for this speaker s. Equations (45) and (46) are respectively a convenient and a more efficient way to compute these statistics. This quantity is symmetric so only the lower triangular part should be stored.

The following statistics are needed in order to update the within-class variances $\mathbf{\Sigma}_i$:

$$\mathbf{S}_i = \sum_{t,j,m} \gamma_{jmi}(t)\mathbf{x}_i(t)\mathbf{x}_i(t)^T. \tag{47}$$

4.4. Accumulation for speaker adaptation

The accumulation required to estimate the speaker vectors $\mathbf{v}^{(s)}$ is analogous to the sub-state vectors accumulation in Equation (41); however, this is a speaker-specific parameter so the accumulation is done separately per speaker . We define a speaker-subspace analogue to the quantity $\mathbf{z}_i(t)$ of (35), which we write as:

$$\mathbf{z}_{jmi}(t) = \mathbf{N}_i^T \ \mathbf{\Sigma}_i^{-1}\mathbf{x}_{jmi}(t), \tag{48}$$

with $\mathbf{x}_{jmi}(t)$ as defined in Equation (43). The statistics are accumulated as follows, where $\mathcal{T}(s)$ is the set of frame indices for speaker s:

$$\gamma_i^{(s)} = \sum_{t\in\mathcal{T}(s),j,m} \gamma_{jmi}(t) \tag{49}$$

$$\mathbf{y}^{(s)} = \sum_{t\in\mathcal{T}(s),i,j,m} \gamma_{jmi}(t)\mathbf{z}_{jmi}(t). \tag{50}$$

Our statistics for CMLLR estimation are:

$$\beta^{(s)} = |\mathcal{T}(s)| \tag{51}$$

$$\mathbf{K}^{(s)} = \sum_{t\in\mathcal{T}(s),j,m,i} \gamma_{jmi}(t)\mathbf{\Sigma}_i^{-1}\boldsymbol{\mu}_{jmi}^{(s)}\mathbf{x}(t)^{+T} \tag{52}$$

$$\mathbf{S}_i^{(s)} = \sum_{t\in\mathcal{T}(s),j,m} \gamma_{jmi}(t)\mathbf{x}(t)^{+}\mathbf{x}(t)^{+T} \tag{53}$$

where $\mathbf{x}^{+} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$ and we need to compute:

$$\boldsymbol{\mu}_{jmi}^{(s)} = \mathbf{M}_i\mathbf{v}_{jm} + \mathbf{o}_i^{(s)} \tag{54}$$

with $\mathbf{o}_i^{(s)}$ as given in Equation (32).

5. Model updates

5.1. Overview

In this section we provide the update formulae for the various parameter types of this model. This is the Maximization step of an E-M process. Derivations are not supplied; see [26], but where they are not obvious we will explain the approach we used. In the following equations we will write $\hat{\mathbf{x}}$ for the newly updated value of parameter $\mathbf{x}$, and just $\mathbf{x}$ for the unchanged one (although in auxiliary functions $\mathbf{x}$ can represent a variable to be solved for). The order of update is not critical and if the update for a parameter type $\mathbf{y}$ refers to the updated value of a different parameter type $\hat{\mathbf{x}}$, if $\mathbf{x}$ has not yet been updated we can just use the unmodified value. Thus, $\hat{\mathbf{x}}$ can be interpreted as the updated value of $\mathbf{x}$, if available. For all parameter types we compute the auxiliary function changes $\Delta\mathcal{Q}$ for diagnostic purposes; these are most useful when summed per parameter type and normalized by dividing by the overall frame count. As mentioned, the E-M theory does not guarantee convergence if all parameter types are updated simultaneously but in our implementation we allow the user to specify any combination of parameter types to be updated on any iteration.

5.2. Sub-state vectors update

Before updating the sub-state vectors $\mathbf{v}_{jm}$ we need to pre-compute the quantities

$$\mathbf{H}_i = \mathbf{M}_i^T \boldsymbol{\Sigma}_i^{-1} \mathbf{M}_i \tag{55}$$

$$\gamma_{jm} = \sum_i \gamma_{jmi}. \tag{56}$$

We write the linear and quadratic terms of the auxiliary function as $\mathbf{g}_{jm}$ and $\mathbf{H}_{jm}$ respectively, and the auxiliary function is as follows[5]:

$$\mathcal{Q}(\mathbf{v}_{jm}) = \mathbf{v}_{jm} \cdot \mathbf{g}_{jm} - \tfrac{1}{2} \mathbf{v}_{jm}^T \mathbf{H}_{jm} \mathbf{v}_{jm} \tag{57}$$

$$\mathbf{g}_{jm} = \mathbf{y}_{jm} + \sum_i \hat{\mathbf{w}}_i \begin{pmatrix} \gamma_{jmi} - \gamma_{jm} \hat{w}_{jmi} \\ + \max(\gamma_{jmi}, \gamma_{jm} \hat{w}_{jmi}) (\hat{\mathbf{w}}_i \cdot \mathbf{v}_{jm}) \end{pmatrix} \tag{58}$$

$$\mathbf{H}_{jm} = \sum_i \gamma_{jmi} \mathbf{H}_i + \max(\gamma_{jmi}, \gamma_{jm} \hat{w}_{jmi}) \hat{\mathbf{w}}_i \hat{\mathbf{w}}_i^T . \tag{59}$$

The notation $\hat{w}_{jmi}$ means the weights computed with the updated values $\hat{\mathbf{w}}_i$ of the $\mathbf{w}_i$ quantities if those were updated before $\mathbf{v}_{jm}$, but in fact we update the vectors $\mathbf{v}_{jm}$ first; in this case the distinction is irrelevant and we could write $\mathbf{w}_i$ and w_{jmi} above. The quantity $\mathbf{v}_{jm}$ on the right of (58) refers to the current (pre-update) value of $\mathbf{v}_{jm}$. Equation (57) should be used to measure the auxiliary function change. The solution $\hat{\mathbf{v}}_{jm} = \mathbf{H}_{jm}^{-1} \mathbf{g}_{jm}$ does not work for singular $\mathbf{H}_{jm}$, so we do

$$\hat{\mathbf{v}}_{jm} = \text{solve\_vec}(\mathbf{H}_{jm}, \mathbf{g}_{jm}, \mathbf{v}_{jm}, K^{\max}) \tag{60}$$

where the function solve_vec is defined in Appendix A; the parameter $K^{\max}$ represents a maximum matrix condition number (e.g. 10 000).

5.3. Sub-state weights estimation

The estimation of the sub-state weights c_{jm} associated with the vectors $\mathbf{v}_{jm}$ is very simple. The auxiliary function is:

$$\mathcal{Q}(\{c_{jm}, 1 \le j \le J, 1 \le m \le M_j\}) = \sum_{j,m} \gamma_{jm} \log c_{jm} \tag{61}$$

with the data count $\gamma_{jm} = \sum_i \gamma_{jmi}$. The sub-state weights must sum to 1 over all m for a particular j, so the update is:

$$\hat{c}_{jm} = \frac{\gamma_{jm}}{\sum_{m=1}^{M_j} \gamma_{jm}}. \tag{62}$$

[5]Most experiments in this paper were run with a previous version of the auxiliary function, with only the second term of the max expression; experiments show no consistent difference except the previous version is subject to a rare instability in specific conditions. Instability is possible because this is a weak-sense auxiliary function in the sense of [25], and unlike the update for $\mathbf{w}_i$ described below we do not check the convergence of the "exact" auxiliary function because we generally believe the contribution from the weight term, which is responsible for the potential instability, is small. Reading Section 5.6 should provide more insight into this issue.

The auxiliary function change should be computed using Equation (61).

5.4. Update for model projections

The auxiliary function for the projection matrix $\mathbf{M}_i$ is:

$$\mathcal{Q}(\mathbf{M}_i) = \mathrm{tr}\left(\mathbf{M}_i^T \boldsymbol{\Sigma}_i^{-1} \mathbf{Y}_i\right) - \tfrac{1}{2}\mathrm{tr}\left(\boldsymbol{\Sigma}_i^{-1} \mathbf{M}_i \mathbf{Q}_i \mathbf{M}_i^T\right) \tag{63}$$

where $\mathbf{Y}_i$ are the statistics accumulated in Equation (42) and

$$\mathbf{Q}_i = \sum_{j,m} \gamma_{jmi} \mathbf{v}_{jm} \mathbf{v}_{jm}^T. \tag{64}$$

If $\mathbf{Q}_i$ is not singular, the solution is $\hat{\mathbf{M}}_i = \mathbf{Y}_i \mathbf{Q}_i^{-1}$, but to cover the general case we should do the update as:

$$\hat{\mathbf{M}}_i = \text{solve\_mat}(\mathbf{Q}_i, \mathbf{Y}_i, \boldsymbol{\Sigma}_i^{-1}, \mathbf{M}_i, K^{\max}) \tag{65}$$

where the function solve_mat is as defined in Appendix A; the maximum matrix condition number $K^{\max}$ can be the same as above (e.g. 10 000).

5.5. Update for speaker projections

The update for the speaker projections $\mathbf{N}_i$ is analogous to the update for the model projections $\mathbf{M}_i$. The auxiliary function is:

$$\mathcal{Q}(\mathbf{N}_i) = \mathrm{tr}\left(\mathbf{N}_i^T \boldsymbol{\Sigma}_i^{-1} \mathbf{Z}_i\right) - \tfrac{1}{2}\mathrm{tr}\left(\boldsymbol{\Sigma}_i^{-1} \mathbf{N}_i \mathbf{R}_i \mathbf{N}_i^T\right) \tag{66}$$

where $\mathbf{Z}_i$ and $\mathbf{R}_i$ are statistics accumulated in Equations (44) and (46); the basic solution is $\hat{\mathbf{N}}_i = \mathbf{Z}_i \mathbf{R}_i^{-1}$ and the "safe" update is:

$$\hat{\mathbf{N}}_i = \text{solve\_mat}(\mathbf{R}_i, \mathbf{Z}_i, \boldsymbol{\Sigma}_i^{-1}, \mathbf{N}_i, K^{\max}). \tag{67}$$

5.6. Update for weight projections

The auxiliary function that we are optimizing is:

$$\mathcal{Q}(\mathbf{w}_1 \ldots \mathbf{w}_I) = \sum_{j,m,i} \gamma_{jmi} \log w_{jmi}. \tag{68}$$

We will not go through the full derivation of the update formulae (see [26, Section 11.7]), but since the derivation for the weight update is not as straightforward as some of the other derivations we will describe the steps involved.

From Equation (7) we can see that $\log w_{jmi} = \mathbf{w}_i^T \mathbf{v}_{jm} - \log \sum_{i'=1}^{I} \exp \mathbf{w}_{i'}^T \mathbf{v}_{jm}$, in which the second term is problematic. We can use the inequality $1 - (x/\bar{x}) \leq -\log(x/\bar{x})$, which is an equality at $x = \bar{x}$, to get rid of the log in the second term and turn it into a sum of terms $\sum_{i'} \exp \mathbf{w}_{i'}^T \mathbf{v}_{jm}$, multiplied by a constant. This step is useful because it removes any correlations in the objective function between different values of i. Then we use a second-order Taylor series approximation to the exponential function to get a quadratic form which is easy to optimize. We make a further modification at this point which is to take a term $\gamma_{jm} w_{jmi}$ which is a weighting term in the quadratic part of the auxiliary function, and replace it with $\max(\gamma_{jmi}, \gamma_{jm} w_{jmi})$. This means that if the value of this term around the unconstrained ML solution would be larger, we take that larger value, and it reflects the heuristic that the weights will probably get closer to the unconstrained ML solution. Taking the larger one is the "safer" option, as this makes the Hessian more negative. We ensure that the resulting auxiliary function still has the same local gradient as (68). By "unconstrained ML solution" we mean the solution we would get if we estimated the weights w_{jmi} as parameters directly.

After the manipulations described above, we can obtain the quadratic auxiliary function and update formula of Equations (70) to (73). This is a "weak-sense" auxiliary function in the sense of [25]. Below, $\mathbf{w}_i^{(p)}$ is the updated weight on iteration p of an iterative process, and we write the weights computed on iteration p as follows:

$$w_{jmi}^{(p)} = \frac{\exp \mathbf{w}_i^{(p)} \cdot \hat{\mathbf{v}}_{jm}}{\sum_{i'=1}^{I} \exp \mathbf{w}_{i'}^{(p)} \cdot \hat{\mathbf{v}}_{jm}}. \tag{69}$$

We use the updated vectors $\hat{\mathbf{v}}_{jm}$ in this calculation, if available. In our experiments we generally used $P_w = 3$ iterations, taking $\mathbf{w}_i^{(0)} = \mathbf{w}_i$ and $\hat{\mathbf{w}}_i = \mathbf{w}_i^{(P_w)}$. The auxiliary function on iteration p for $1 \leq p \leq P_w$ is:

$$\begin{aligned} \mathcal{Q}(\mathbf{w}_i^{(p)}) &= (\mathbf{w}_i^{(p)} - \mathbf{w}_i^{(p-1)}) \cdot \mathbf{g}_i^{(p)} \\ &\quad -\tfrac{1}{2} (\mathbf{w}_i^{(p)} - \mathbf{w}_i^{(p-1)})^T \mathbf{F}_i^{(p)} (\mathbf{w}_i^{(p)} - \mathbf{w}_i^{(p-1)}) \qquad (70) \\ \mathbf{g}_i^{(p)} &= \sum_{j,m} (\gamma_{jmi} - \gamma_{jm} w_{jmi}^{(p-1)}) \hat{\mathbf{v}}_{jm} \qquad (71) \\ \mathbf{F}_i^{(p)} &= \sum_{j,m} \max(\gamma_{jmi}, \gamma_{jm} w_{jmi}^{(p-1)}) \hat{\mathbf{v}}_{jm} \hat{\mathbf{v}}_{jm}^T. \qquad (72) \end{aligned}$$

The basic solution is $\mathbf{w}_i^{(p)} = \mathbf{w}_i^{(p-1)} + \mathbf{F}_i^{(p)^{-1}} \mathbf{g}_i^{(p)}$, and the "safe" solution is:

$$\mathbf{w}_i^{(p)} = \mathbf{w}_i^{(p-1)} + \text{solve\_vec}(\mathbf{F}_i^{(p)}, \mathbf{g}_i^{(p)}, \mathbf{0}, K^{\max}). \tag{73}$$

However, we need to modify this process to ensure convergence. Maximizing (70) is not guaranteed to increase the "exact" auxiliary function of (68) so on each iteration we need to check whether (68) decreased and if so decrease the step size (e.g., go halfway back to $\mathbf{w}_i^{(p-1)}$).

There is a choice as to whether we do the update of Equation (73) and the subsequent step of checking the auxiliary function, sequentially or in parallel. In the sequential version, we do the update for each i in turn; in the parallel one we do them all at the same time (this does not relate to parallel processing). The parallel version is simpler and easier to implement but the sequential version seems likely to be safer in preventing parameter overshoot. It is the parallel version that corresponds to Equations (70) to (73). Our experiments on the difference between the parallel and sequential versions have been inconclusive. Because the performance of the model seems to be quite sensitive to the weight update process we give both methods as Algorithms 5.1 and 5.2. In the sequential version, the normalizer k_{jm} and quantities added to or subtracted from it should be stored as log values, and the algorithm requires a "log subtract" function similar to the normal "log add" function. We recommend implementing a "log" version of $a + b - c$ by first computing the absolute difference between b and c and then adding or subtracting it from a as appropriate. Care should be taken that round-off errors do not cause the **while** loop to fail to terminate (e.g. introduce a maximum number of iterations).

The methods used here to derive the weight update were also used to derive the update for the vectors $\mathbf{v}_{jm}$ in Section 5.2 because the vectors also affect the weights w_{jmi}; however, in the updates described in Section 5.2 we do not check convergence of the "exact" auxiliary function because we believe that the auxiliary function for the vectors is dominated by terms arising from the mean.

Bearing in mind the complexity of the methods described here, one can ask whether a simpler method such as conjugate gradient descent might suffice. One reason is that for the update of the vectors $\mathbf{v}_{jm}$ (Section 5.2), a method based on auxiliary functions is easier to integrate with the other part of the auxiliary function. However for $\mathbf{w}_i$ it is quite possible that a simpler and equally effective method using conjugate gradient methods could

be devised.

Algorithm 5.1 Weight update: sequential version

$\forall i,\ \hat{\mathbf{w}}_i \leftarrow \mathbf{w}_i$
$\forall (j,m),\ k_{jm} \leftarrow \sum_i \exp(\hat{\mathbf{w}}_i \cdot \hat{\mathbf{v}}_{jm})$ // store as log
for $p \leftarrow 1 \ldots P_w$ **do**
  for $i \leftarrow 1 \ldots I$ **do**
    $\hat{\mathbf{w}}^{\text{tmp}} \leftarrow \hat{\mathbf{w}}_i$
    Using $\hat{w}_{jmi} = \exp(\hat{\mathbf{w}}_i \cdot \hat{\mathbf{v}}_{jm} - \log k_{jm})$,
    $\mathbf{g} \leftarrow \sum_{j,m} (\gamma_{jmi} - \gamma_{jm}\hat{w}_{jmi})\hat{\mathbf{v}}_{jm}$
    $\mathbf{F} \leftarrow \sum_{j,m} \max(\gamma_{jmi}, \gamma_{jm}\hat{w}_{jmi})\hat{\mathbf{v}}_{jm}\hat{\mathbf{v}}_{jm}^T$
    $\hat{\mathbf{w}}_i \leftarrow \hat{\mathbf{w}}_i + \text{solve\_vec}(\mathbf{F}, \mathbf{g}, \mathbf{0}, K^{\max})$
    while $\begin{pmatrix} \sum_{j,m} \gamma_{jmi} (\hat{\mathbf{w}}_i - \hat{\mathbf{w}}^{\text{tmp}}) \cdot \hat{\mathbf{v}}_{jm} \\ -\gamma_{jm} \log \begin{pmatrix} 1 + \exp(\hat{\mathbf{w}}_i \cdot \hat{\mathbf{v}}_{jm} - k_{jm}) \\ -\exp(\hat{\mathbf{w}}^{\text{tmp}} \cdot \hat{\mathbf{v}}_{jm} - k_{jm}) \end{pmatrix} \end{pmatrix} < 0$ **do**
      $\hat{\mathbf{w}}_i \leftarrow \frac{1}{2}(\hat{\mathbf{w}}^{\text{tmp}} + \hat{\mathbf{w}}_i)$
    end while
    $\forall (j,m),\ k_{jm} \leftarrow k_{jm} + \exp(\hat{\mathbf{w}}_i \cdot \hat{\mathbf{v}}_{jm}) - \exp(\hat{\mathbf{w}}^{\text{tmp}} \cdot \hat{\mathbf{v}}_{jm})$
  end for
end for

5.7. Update for within-class covariances

The update for $\mathbf{\Sigma}_i$ (prior to flooring) is:

$$\mathbf{S}_i^{\text{means}} = \sum_{j,m} \gamma_{jmi} \boldsymbol{\mu}_{jmi} \boldsymbol{\mu}_{jmi}^T \tag{74}$$

$$\mathbf{\Sigma}_i^{\text{ml}} = (1/\gamma_i) \left(\mathbf{S}_i + \mathbf{S}_i^{\text{means}} - \mathbf{Y}_i \mathbf{M}_i^T - \mathbf{M}_i \mathbf{Y}_i^T \right) \tag{75}$$

where the covariance statistics $\mathbf{S}_i$ were accumulated in (47) and $\gamma_i = \sum_{j,m} \gamma_{jmi}$. Note that (74),(75) do not refer to any updated quantities. As with conventional systems, variance flooring is helpful. The floor used is $f\mathbf{\Sigma}^{\text{avg}}$, where $\mathbf{\Sigma}^{\text{avg}} = (\sum_i \gamma_i \mathbf{\Sigma}_i^{\text{ml}}) / \sum_i \gamma_i$ and e.g. $f = 0.2$ is a reasonable setting. We set $\hat{\mathbf{\Sigma}}_i \leftarrow \text{floor}(\mathbf{\Sigma}_i^{\text{ml}}, f\mathbf{\Sigma}^{\text{avg}})$ using the floor function defined in Algorithm 5.3.

The auxiliary function change for the variance $\mathbf{\Sigma}_i$ can be calculated as:

$$\Delta\mathcal{Q}(\mathbf{\Sigma}_i) = -\tfrac{\gamma_i}{2} \left(\log\det \hat{\mathbf{\Sigma}}_i - \log\det \mathbf{\Sigma}_i + \text{tr}\left(\hat{\mathbf{\Sigma}}_i^{-1} \mathbf{\Sigma}_i^{\text{ml}}\right) - \text{tr}\left(\mathbf{\Sigma}_i^{-1} \mathbf{\Sigma}_i^{\text{ml}}\right) \right). \tag{76}$$

Algorithm 5.2 Weight update: "parallel" version

$\forall i, \hat{\mathbf{w}}_i \leftarrow \mathbf{w}_i$
$\forall (j,m,i),\ \hat{w}_{jmi} \leftarrow \frac{\exp(\hat{\mathbf{w}}_i \cdot \hat{\mathbf{v}}_{jm})}{\sum_{i'} \exp(\hat{\mathbf{w}}_{i'} \cdot \hat{\mathbf{v}}_{jm})}$
for $p \leftarrow 1 \dots P_w$ **do**
  $a \leftarrow \sum_{j,m,i} \gamma_{jmi} \log \hat{w}_{jmi}$
  for $i \leftarrow 1 \dots I$ **do**
    $\hat{\mathbf{w}}_i^{\text{tmp}} \leftarrow \hat{\mathbf{w}}_i$
    $\mathbf{g} \leftarrow \sum_{j,m} (\gamma_{jmi} - \gamma_{jm} \hat{w}_{jmi}) \hat{\mathbf{v}}_{jm}$
    $\mathbf{F} \leftarrow \sum_{j,m} \max(\gamma_{jmi}, \gamma_{jm} \hat{w}_{jmi}) \hat{\mathbf{v}}_{jm} \hat{\mathbf{v}}_{jm}^T$
    $\hat{\mathbf{w}}_i \leftarrow \hat{\mathbf{w}}_i + \text{solve\_vec}(\mathbf{F}, \mathbf{g}, \mathbf{0}, K^{\max})$
  end for
  $\forall (j,m,i),\ \hat{w}_{jmi} \leftarrow \frac{\exp(\hat{\mathbf{w}}_i \cdot \hat{\mathbf{v}}_{jm})}{\sum_{i'} \exp(\hat{\mathbf{w}}_{i'} \cdot \hat{\mathbf{v}}_{jm})}$
  while $\sum_{j,m,i} \gamma_{jmi} \log \hat{w}_{jmi} < a$ **do**
    $\forall i,\ \hat{\mathbf{w}}_i \leftarrow \frac{1}{2}(\hat{\mathbf{w}}_i^{\text{tmp}} + \hat{\mathbf{w}}_i)$
    $\forall (j,m,i), \hat{w}_{jmi} \leftarrow \frac{\exp(\hat{\mathbf{w}}_i \cdot \hat{\mathbf{v}}_{jm})}{\sum_{i'} \exp(\hat{\mathbf{w}}_{i'} \cdot \hat{\mathbf{v}}_{jm})}$
  end while
end for

Algorithm 5.3 $\tilde{\mathbf{S}} = \text{floor}(\mathbf{S}, \mathbf{F})$

Require: $\mathbf{F}$ symmetric positive definite
Require: $\mathbf{S}$ symmetric positive semi-definite
$\mathbf{F} = \mathbf{L}\mathbf{L}^T$ (Cholesky decomposition)
$\mathbf{T} \leftarrow \mathbf{L}^{-1}\mathbf{S}\mathbf{L}^{-T}$
$\mathbf{T} = \mathbf{U}\mathbf{D}\mathbf{V}^T$ (Singular Value Decomposition)
Set diagonal matrix $\tilde{\mathbf{D}}$ to $\mathbf{D}$ floored to 1, i.e. $\tilde{d}_{ii} = \max(d_{ii}, 1)$.
$\tilde{\mathbf{T}} \leftarrow \mathbf{U}\tilde{\mathbf{D}}\mathbf{U}^T$ // Important to use the same matrix (e.g. $\mathbf{U}$) on left and right.
$\tilde{\mathbf{S}} \leftarrow \mathbf{L}\tilde{\mathbf{T}}\mathbf{L}^T$

5.8. Sub-state splitting

If sub-states are to be used, it is necessary to split sub-states to reach the desired number. We set a target overall number of sub-states N on each iteration, and work out from this a target number of sub-states for each state j. The way we do this is similar to HTK's `PS` command in `HHEd`[6]; we assign the sub-states proportional to a small power p, typically 0.2, of the data count for that state. Thus we have a target $N(j) = \max(1, \lfloor \alpha \gamma_j^p + \frac{1}{2} \rfloor)$, where $\gamma_j = \sum_{m,i} \gamma_{jmi}$ is the total count for state j, and α is set so that $\sum_j N(j)$ is close to the target N; $\alpha = N / \sum_j \gamma_j^p$ is a reasonable approximation. Choosing which set of sub-states m to split within a state j can be based on highest count γ_{jm}. Prior to splitting we compute a matrix $\mathbf{H}^{(\mathrm{sm})}$ and its Cholesky factor $\mathbf{G}$ as follows:

$$\mathbf{H}^{(\mathrm{sm})} = \frac{1}{\sum_i \gamma_i} \sum_i \gamma_i \mathbf{H}_i \tag{77}$$

$$\mathbf{H}^{(\mathrm{sm})} = \mathbf{G}\mathbf{G}^T \text{ (Cholesky decomposition)} \tag{78}$$

where $\mathbf{H}_i$ is as defined in (55) and $\gamma_i = \sum_{j,m} \gamma_{jmi}$. $\mathbf{H}^{(\mathrm{sm})}$ is analogous to an averaged inverse variance on the vectors $\mathbf{v}_{jm}$, and we use it to provide a natural scaling. When we split a sub-state m we divide the weight c_{jm} in two, and perturb the resulting two vectors by

$$\pm 0.1 \mathbf{G}^{-1} \mathbf{r} \tag{79}$$

where $\mathbf{r}$ is a random normally distributed vector (e.g., obtained using the Box-Muller method), drawn independently for each split. Compare with the Gaussian mixture splitting approach used in HTK [39].

5.9. Increasing the phonetic or speaker-subspace dimension

If the desired phonetic dimension S is larger than $D+1$, it is necessary to train the model for some iterations and then increase S by some amount no larger than D. The same is true of the speaker-subspace dimension T. If we were increasing S to S' or T to T', we would do respectively:

$$\mathbf{M}_i \leftarrow [\mathbf{M}_i \;\; \mathbf{j}_1 \ldots \mathbf{j}_{S'-S}] \tag{80}$$

$$\mathbf{N}_i \leftarrow [\mathbf{N}_i \;\; \mathbf{j}_1 \ldots \mathbf{j}_{T'-T}] \tag{81}$$

[6] Not documented in current HTK Book

where $\mathbf{j}_d$ is the d'th column of the matrix $\mathbf{J}$ computed in (23). The sub-state vectors $\mathbf{v}_{jm}$ and the weight projections $\mathbf{w}_i$ would be extended by appending $S' - S$ zeros. If the speaker vectors $\mathbf{v}^{(s)}$ were being stored between iterations rather than being computed afresh each time we would also append $T' - T$ zeros to the speaker vectors $\mathbf{v}^{(s)}$.

5.10. Update for speaker vectors

The update for the speaker vectors $\mathbf{v}^{(s)}$ does not take place at the Maximization step of the overall E-M process, but as part of a separate E-M process specific to the speaker. For this update, we use the count and linear statistics $\gamma_i^{(s)}$ and $\mathbf{y}^{(s)}$ accumulated in (49) and (50). We should pre-compute the quantities

$$\mathbf{H}_i^{\text{spk}} = \mathbf{N}_i^T \mathbf{\Sigma}_i^{-1} \mathbf{N}_i \tag{82}$$

which are speaker-subspace versions of (55). The auxiliary function is:

$$\mathcal{Q}(\mathbf{v}^{(s)}) = \mathbf{v}^{(s)} \cdot \mathbf{y}^{(s)} - \tfrac{1}{2} \mathbf{v}^{(s)^T} \mathbf{H}^{(s)} \mathbf{v}^{(s)} \tag{83}$$

$$\mathbf{H}^{(s)} = \sum_i \gamma_i^{(s)} \mathbf{H}_i^{\text{spk}}. \tag{84}$$

The solution is $\hat{\mathbf{v}}^{(s)} = \mathbf{H}^{(s)^{-1}} \mathbf{y}^{(s)}$ if $\mathbf{H}^{(s)}$ is invertible, and to cover the general case we do:

$$\hat{\mathbf{v}}^{(s)} = \text{solve\_vec}(\mathbf{H}^{(s)}, \mathbf{y}^{(s)}, \mathbf{v}^{(s)}, K^{\max}). \tag{85}$$

In our implementation, during training we recompute $\mathbf{v}^{(s)}$ on each iteration of overall E-M, starting from $\mathbf{v}^{(s)} = \mathbf{0}$ each time and using just one iteration of E-M per speaker. During testing we use either one or two iterations of E-M, again starting from $\mathbf{v}^{(s)} = \mathbf{0}$.

5.11. Update for Constrained MLLR: Overview

Here we provide an overview of the CMLLR update. The details are in Appendix Appendix B; here we aim to give some idea of the structure of the computation. We do not use the standard row-by-row update method [8] because it only applies to diagonal models. There are extensions that work for full-covariance models [35, 31] but the method we use here is more efficient. We optimize the matrix by repeated line-search in a preconditioned gradient direction. There are two pre-computation tasks that must be done at training time. The first phase of pre-computation requires as input the

SGMM and per-state data counts γ_j, and it outputs a normalizing transform $\mathbf{W}_{\text{pre}}$ with the same structure as a CMLLR matrix, together with a diagonal matrix $\mathbf{D}$ that corresponds to eigenvalues in an LDA problem. The second task is to train the basis of Equation (12), if needed, and this requires statistics accumulated from the training data; the problem is like PCA in dimension $D(D+1)$. In test time, our computation is an iterative procedure like the standard row-by-row update; it requires the pre-computed quantities together with the statistics from Equations (51) to (53).

6. Training schedule

In this section we provide further details on our overall training procedure, and explain our schedule for updating parameters. Before training our UBM and SGMM, we assume that some kind of baseline system (e.g. a GMM system) has been supplied; we call this Λ^{prev}. This also supplies the phone context tying (decision trees) for use in the SGMM system. The initialization of the UBM and SGMM is as described in Section 2.3. Before explaining the detailed E-M schedule, we first summarize the accumulation and update phases of the E-M process.

Algorithm 6.1 shows the accumulation process, assuming we are using speaker vectors; $N^{\text{old-align}}$ represents the number of initial iterations on which we align with the old (GMM) system. This algorithm mainly shows that we need to estimate the speaker vectors for the speaker before accumulating the main statistics. The types of statistics we accumulate will be dictated by a set of flags on each iteration. Algorithm 6.2 shows the update phase. The main point of showing this is to specify the order of update we use and to show the other tasks that we do in the update phase, such as splitting sub-states.

Table 4 shows the training schedule we used in our experiments. We train the system in "epochs" of 8 iterations each. In the system described in the table, speaker vectors were used, but we did not use CMLLR in training time. We update $\mathbf{M}_i$ and $\mathbf{N}_i$ on alternate iterations. Even when not using the speaker subspace, we still typically update $\mathbf{M}_i$ only on every other iteration. Using eight iterations between each phase of sub-state splitting was probably not necessary but was done in order to accurately test the word error rate at each phase. We used $N^{\text{old-align}} = 8$, i.e. we aligned using the baseline HTK model for the first epoch. The table does not mention estimation of the CMLLR pre-transform (Appendix B.2) or basis (Appendix B.3).

Algorithm 6.1 Accumulation for speaker s on iteration n

if $T > 0$ (speaker subspace set up) **then**
  $\mathbf{v}^{(s)} \leftarrow \mathbf{0}$
  for each utterance of this speaker **do**
    Obtain Viterbi alignment $\gamma_j(t)$ from our model Λ, or from Λ^{prev} if $n < N^{\text{old-align}}$.
    Accumulate speaker-vectors statistics $\mathbf{y}^{(s)}$ and $\gamma_i^{(s)}$
  end for
  Compute $\mathbf{v}^{(s)}$ (Section 5.10)
end if
for each utterance of this speaker **do**
  Obtain Viterbi alignment $\gamma_j(t)$ from our model Λ, or from Λ^{prev} if $n < N^{\text{old-align}}$.
  Accumulate statistics for selected subset of parameters, see Section 4
end for
if computing CMLLR basis on this iteration **then**
  Do accumulation for second CMLLR computation phase; see Appendix B.3, Equation (B.10).
end if

Algorithm 6.2 Update phase for SGMMs

Do some specified subset of the E-M updates below:
  Update vectors $\mathbf{v}_{jm}$ // Section 5.2.
  Update sub-state weights c_{jm} // Section 5.3.
  Update projections $\mathbf{M}_i$ // Section 5.4.
  Update speaker projections $\mathbf{N}_i$ // Section 5.5.
  Update weight projections $\mathbf{w}_i$ // Section 5.6.
  Update covariance matrices $\mathbf{\Sigma}_i$ // Section 5.7.
If requested on this iteration, do some subset of the following tasks:
  Split sub-states // Section 5.8
  Increase the dimensions S or T // Section 5.9
  Estimate the pre-transform for CMLLR // Appendix B.2
  Estimate the CMLLR basis // Appendix B.3
Recompute normalizers n_{jmi} // Section 3.2

Table 4: Typical training schedule

Epoch	Iteration			Align
	First (1)	Even (2,4,6,8)	Odd (3,5,7)	
	Initialize with $S = 40$, $T = 0$			
1	$\mathbf{v}$	$\mathbf{v}, \mathbf{M}, \mathbf{\Sigma}, \mathbf{w}$	$\mathbf{v}, \mathbf{M}, \mathbf{\Sigma}, \mathbf{w}$	Prev
2	$\mathbf{v}, \mathbf{\Sigma}, \mathbf{w}$	$\mathbf{v}, \mathbf{M}, \mathbf{\Sigma}, \mathbf{w}$	$\mathbf{v}, \mathbf{\Sigma}, \mathbf{w}$	Self
3	$\mathbf{v}, \mathbf{\Sigma}, \mathbf{w}$ Split: 2700 sub-states Increase T to 39	$\mathbf{v}, c, \mathbf{M}, \mathbf{\Sigma}, \mathbf{w}$	$\mathbf{v}, c, \mathbf{N}, \mathbf{\Sigma}, \mathbf{w}$	
4	$\mathbf{v}, c, \mathbf{N}, \mathbf{\Sigma}, \mathbf{w}$ Split: 4000 sub-states	$\mathbf{v}, c, \mathbf{M}, \mathbf{\Sigma}, \mathbf{w}$	$\mathbf{v}, c, \mathbf{N}, \mathbf{\Sigma}, \mathbf{w}$	
5	$\mathbf{v}, c, \mathbf{N}, \mathbf{\Sigma}, \mathbf{w}$ Split: 6000 sub-states	$\mathbf{v}, c, \mathbf{M}, \mathbf{\Sigma}, \mathbf{w}$	$\mathbf{v}, c, \mathbf{N}, \mathbf{\Sigma}, \mathbf{w}$	
6	$\mathbf{v}, c, \mathbf{N}, \mathbf{\Sigma}, \mathbf{w}$ Split: 9000 sub-states	$\mathbf{v}, c, \mathbf{M}, \mathbf{\Sigma}, \mathbf{w}$	$\mathbf{v}, c, \mathbf{N}, \mathbf{\Sigma}, \mathbf{w}$	
7	$\mathbf{v}, c, \mathbf{N}, \mathbf{\Sigma}, \mathbf{w}$ Split: 12000 sub-states	$\mathbf{v}, c, \mathbf{M}, \mathbf{\Sigma}, \mathbf{w}$	$\mathbf{v}, c, \mathbf{N}, \mathbf{\Sigma}, \mathbf{w}$	
8	$\mathbf{v}, c, \mathbf{N}, \mathbf{\Sigma}, \mathbf{w}$ Split: 16000 sub-states	$\mathbf{v}, c, \mathbf{M}, \mathbf{\Sigma}, \mathbf{w}$	$\mathbf{v}, c, \mathbf{N}, \mathbf{\Sigma}, \mathbf{w}$	

In most experiments, we did not use the CMLLR basis and in those cases we estimated the pre-transform from the model in test time if needed for adaptation. In order to make this possible without loading statistics, we stored the per-state counts γ_j with the model. In experiments where we used the CMLLR basis, we computed the pre-transform and basis on each training epoch.

7. Experimental setup

7.1. Software environment

Experiments were conducted in a newly written `C++` framework but leveraging a large amount of existing freely available code. We used HTK [39] to extract features and build the baseline models. We also made use of the HTK-generated phonetic context decision trees. We used the OpenFST [1] Weighted Finite State Transducer [24] library and command line tools to simplify the tasks of training and decoding. Language models (LMs) were built using the SRILM tools [37]. Our SGMM training and evaluation code makes heavy use of matrix operations, and for this we created a `C++` wrapper

for standard C-based matrix and vector libraries. We use the ATLAS and CLAPACK libraries and some code derived from the TNT library. Work is ongoing to prepare our code for public release; please contact the authors for availability.

7.2. Data sets and baseline systems

We report experiments on the CallHome English, Spanish and German databases. The CallHome corpora were collected by the Linguistic Data Consortium (LDC) for those languages and also for Arabic, Mandarin and Japanese. We chose CallHome because of the manageable size of the databases, which led to fast turnaround time in experiments, and because multiple languages were available in a common format. The conversational nature of the CallHome data along with high out-of-vocabulary rates, use of foreign words and telephone channel distortions make speech recognition on these databases challenging.

7.2.1. English system

The English CallHome database [5] consists of 120 spontaneous telephone conversations between native English speakers. Eighty conversations, corresponding to about 15 hours of speech, are used as training data. Two sets of 20 conversations, roughly containing 1.8 hours of speech each, form the test and development sets. We use 39 dimensional PLP [18] features including delta and delta-delta parameters. Using HTK we built standard 3-state left-to-right cross-word context dependent baseline systems, with and without Speaker Adaptive Training (SAT), and tested with and without CMLLR. Our lexicon contains 62K words, with an OOV rate of 0.4% on the test data. Our trigram language model was built using the SRILM tools [37], and has 3.5 million n-grams in total with a perplexity of 95. It is interpolated from individual models created from the English CallHome training data, the Switchboard corpus [13], the Gigaword corpus [17], and 100 million words of web data. Weights are trained on the development test set. The web data was obtained by searching the web for sentences containing high frequency bigrams and trigrams occurring in the training text of the CallHome corpus. We also built a bigram language model on the same data, for faster decoding turnaround and less memory usage. The 90K PRONLEX dictionary [20] with 47 phones is used as the lexicon; this is supplied with the CallHome corpus by the LDC.

7.2.2. Spanish and German systems

Similar to the English database, the CallHome Spanish and German databases [7, 6] consist of 120 and 100 spontaneous telephone conversations respectively between native speakers. This gives 16 and 15 hours of speech respectively in the training sets. Two sets of 20 conversations, containing 2 and 3.7 hours of speech respectively, form the test sets for the Spanish and German systems. The word-list for Spanish contains 45K words, and the language model is trained on the Spanish CallHome transcripts and 20 million words of web data, collected as above; the interpolation weight was estimated on the test data in this case. The language model had 1.4 million n-grams in total and a perplexity of 127. The Spanish pronunciation dictionary, provided by the LDC for the CallHome corpus, was automatically generated from letter to sound rules. The features and type of system are as for English. We did not create a word-level decoding setup for the German system, due to time limitations. Instead we restricted ourselves to phone decoding for German.

7.3. Decoding setup

We built our own recognizer, called `Kaldi`, and used the OpenFST tools to build a recognition network from a language model, a lexicon and a set of phonetic context decision trees. `Kaldi` can read in HTK models as well as SGMMs. We also used the HTK recognizer `HDecode` to compare with `Kaldi`. Recognition experiments show the two decoders to be within half a percent WER of each other, with neither having a consistent edge. We show baseline results with `HDecode` and SGMM results with `Kaldi`. English word recognition experiments are scored with the NIST scoring tool `sclite`; all other scoring is with HTK's `HResults`. To include results from German, we built for all languages a phone bigram language model and tested them with this, measuring the Phone Error Rate (PER). We refer to this as a "phonotactic" system; the term is used in the language identification literature.

8. Results

8.1. Speaker independent single-language experiments

First we discuss experiments on basic systems in English, Spanish and German without speaker adaptation, where we compare baseline and SGMM systems of different sizes.

The English system has 1920 tied states. We decoded the baseline system with a language model scale of 13 and an acoustic penalty of 10, which was optimized on the test set; we did the same for the SGMM, and this resulted in a language model scale of 10 and an acoustic penalty of 5 (or zero in some experiments). This is consistent with previous experience [33] that the SGMM requires less extreme scaling. We show, in Table 5, baseline HTK results with a varying number of Gaussians per state. The best result, at 52.3% WER, is similar to a previously published error rate of 53.7% WER in [41] which appears to be fully adapted and discriminatively trained.

The comparable SGMM results are shown in Table 6. This system was initialized with, and used the phonetic context tying of, the HTK system with 1920 tied states and 16 Gaussians per state. We used $I = 400$ Gaussians in the UBM, and subspace dimension $S = 40$. The different lines of Table 6 show results at various epochs of training, each corresponding to a line of Table 4 which shows the system building procedure (however, we did no speaker adaptation in this experiment). It can be seen that even the most basic SGMM system without sub-states (top two lines), does better than the best GMM system. After adding sub-states, the Word Error Rate improvement is 4.8% absolute (9.2% relative) versus the HTK baseline. The numbers in bold are the best word error rates in the column, for easy comparison between different experiments.

Tables 5 and 6 also show similar results for a Spanish language system. It had 1584 tied states. The baseline WER, at 68.1%, is somewhat worse than a previously published WER of 64.1% without adaptation [41]; we suspect that deficiencies in text normalization may have led to sub-par results. Again the SGMM approach gives improvement, with WER decreasing from 68.1% to 65.2% (2.9% absolute, or 4.3% relative). We also ran a similar experiment (not shown in the table) with a larger number of Gaussians $I = 800$, and the best WER obtained was almost the same, at 65.3%. This was obtained with a smaller number of sub-states (1400), so the total number of parameters was similar to that for the best result in Table 6.

In experiments described below, the training schedule in terms of the number of sub-states on each epoch is always the same as described here, so in some tables we will report only the epoch number and not the number of sub-states.

Table 5: Baseline HTK systems – English and Spanish, Trigram LM

#Gauss /state	English #Params	English %WER	Spanish #Params	Spanish %WER
12	1.8M	53.2	1.5M	69.1
14	2.1M	52.6	1.8M	68.8
16	2.4M	**52.3**	2.0M	68.5
18	2.7M	52.5	2.3M	68.5
20	3.0M	52.4	2.5M	68.3
22	3.3M	52.7	2.8M	**68.1**
24	3.6M	52.9	3.0M	68.1

Table 6: SGMM system ($I = 400$) – English and Spanish, Trigram LM

Epoch	#Sub-states (English/Spanish)	#Params (English/Spanish)	%WER English	%WER Spanish
1	1921/1584	1.01M/1.00M	50.0	66.4
2	1921/1584	1.01M/1.00M	48.9	65.9
3	2.7K	1.05M	48.5	65.7
4	4K	1.10M	48.5	65.8
5	6K	1.18M	48.1	65.9
6	9K	1.31M	47.8	65.7
7	12K	1.43M	**47.5**	65.7
8	16K	1.59M	47.7	65.3
9	22K	1.83M	47.6	**65.2**

8.2. Multilingual experiments

With the SGMM we are able to train multilingual systems in a unique way, maintaining completely separate sets of acoustic states between languages but sharing the parameters $\mathbf{M}_i$, $\mathbf{w}_i$ and $\mathbf{\Sigma}_i$. In contrast, the traditional approach to multilingual speech recognition, e.g. see [34], is based on a shared phone-set. We do not show any baseline multilingual approaches other than the single-language Maximum Likelihood baseline. This is because our understanding of the literature is that data from other languages does not in general reduce the Word Error Rate of the target language (e.g., [23]); in addition, the methods used are typically quite complicated. There is no simple technique such as MAP adaptation that would apply in the cross language case, because in general languages have distinct phone sets and there

is no very natural way to establish a correspondence between phones across languages.

We trained a system in which the UBM and the shared parameters were shared across multiple languages (English, Spanish and German). The phone sets were completely distinct, i.e. we did not merge any phones across languages. We initialized the systems with a UBM derived from clustering the Spanish model and trained the UBM on the merged data, after experiments on the English system found the Spanish UBM to be 0.5% absolute better than the English UBM for initializing the system. Thereafter we trained as if we had a single language, on an utterance list that included all three languages. The results are shown in Table 7. The training schedule was as before but with 16 iterations per epoch. Comparing with the monolingual SGMM baseline (Table 6), the English system improved from 47.5% WER to 44.9% (5.5% relative), and the Spanish system improved from 65.2% to 64.4% (1.2% relative).

In order to show results on German in the absence of a word decoding setup, we repeated the same experiments with phoneme recognition, using a "phonotactic" language model (i.e. a phone bigram), trained on the acoustic training transcripts. The systems were the same ones tested above. Table 8 shows the best baseline HTK system, the best monolingual SGMM system, and the best multilingual-trained results. The Phone Error Rate (PER) results on German are consistent with the Spanish and English results, with the SGMM system improving over the baseline and the multilingual system giving further improvements.

8.3. Multilingual experiments with one hour of English

In order to investigate speech recognition performance when the amount of training data is very limited, we did experiments with a 1 hour subset of the 15 hour English CallHome training database. To form this subset, we picked segments randomly from all training conversations. Using HTK, we built three different phonetic context trees with about 500, 1000 and 1500 clustered states respectively. For each configuration we picked the best tree. This resulted in a tree size of 500 for the baseline HTK result, 1000 for the SGMM result and 1500 for the multilingual SGMM result. The HTK baseline system had a 70.5% error rate on this data. When we built an SGMM system on the same amount of data, using $I = 400$ and a subspace dimension $S = 20$, the best word error rate (obtained after the first epoch of training) was 67.8%. When we built a multilingual system with the shared parameters

Table 7: Multilingual experiments– English and Spanish, Trigram LM

Epoch	English		Spanish	
	#Sub-states	%WER	#Sub-states	%WER
1	1921	50.7	1582	66.7
2	1921	49.2	1582	66.4
3	1921	48.8	1582	66.4
4	1921	48.5	1582	66.4
5	2134	48.5	1909	66.1
6	3240	47.5	2823	65.2
7	4314	46.4	3738	65.1
8	5835	46.3	4891	64.7
9	7992	45.8	6779	64.6
10	11K	45.9	9.3K	64.5
11	14K	45.6	12K	**64.4**
12	19K	**44.9**	16K	64.7
13	25K	45.4		
14	33K	45.5		

Table 8: Phone decoding – English, Spanish and German, %PER

	English	Spanish	German
HTK	54.9	46.2	56.3
SGMM	51.7	44.0	53.4
+multilingual	50.2	43.5	52.4

trained also on Spanish and German, the best word error rate, achieved after the second epoch of training, was 59.2%– a relative improvement of 16% versus the HTK baseline[7]. This shows that the SGMM approach can lead to particularly large improvements when training on small data-sets, as long as we have some less relevant data available to train the shared parameters.

[7]We were able to obtain approximately 2% absolute better WER than this when using the original system built on 15 hours of data to obtain the Viterbi alignments used in training. This fact is not relevant to the scenario of limited training data but does show the importance of good training-data alignments.

8.4. Importance of different model features

We did experiments to investigate the importance of different aspects of our modeling strategy, by disabling the re-estimation of certain parameter types and also by training a diagonal-covariance SGMM system. The results are in Table 9. A basic SGMM system is contrasted with a system where the variances $\mathbf{\Sigma}_i$ are constrained to be diagonal (the "Diag" column), and systems in which we disable the update of various parameters: the variances $\mathbf{\Sigma}_i$, the weight projections $\mathbf{w}_i$ and the mean projections $\mathbf{M}_i$. These parameters are left at their initial values. The baseline HTK system has a WER of 54.7% with the same (bigram) language model. In cases where the modification seems to significantly affect the ratio of insertions to deletions, we re-tune the language model scale on the eighth iteration. The worst system is the one in which the update of the weight projections $\mathbf{w}_i$ is disabled; this shows the importance of the mixture-weight parameters in the system. We also get a large degradation by using diagonal variances. The change that makes the least difference is leaving the $\mathbf{\Sigma}_i$ at the initial values $\bar{\mathbf{\Sigma}}_i$ (the UBM variances).

Table 9: SGMM, disabling certain system features – English %WER, Bigram LM

Epoch	#Sub-states	SGMM	Diag	Disable-update		
				$\mathbf{\Sigma}_i$	$\mathbf{w}_i$	$\mathbf{M}_i$
1	1921	52.5	56.0	54.3	63.4	62.5
2	1921	51.4	55.6	53.1	60.6	60.7
3	2K	50.9	55.1	52.5	59.8	59.6
4	4K	50.5	54.3	52.4	50.9	57.9
5	6K	50.0	54.1	52.1	57.8	56.5
6	9K	49.6	53.9	51.9	57.0	55.0
7	12K	**49.1**	**53.8**	51.5	56.7	53.7
8	16K	49.3		**51.3**	**56.1**	52.9
9	20K					**52.3**
8	(+tune LM-scale)		**53.5** @11.0	**50.7** @8.0	**55.7** @8.0	

8.5. Adaptation with Constrained MLLR

Table 10 shows the baseline HTK system with and without adaptation, and in various stages of the Speaker Adaptive Training (SAT) process. We show these results for a system with 18 Gaussians per state because this was

Table 10: English– baseline adaptation results, %WER (2-class)

SAT Epochs	Bigram # Iters CMLLR 0	1	4	Trigram # Iters CMLLR 0	1	4
0	55.8	53.9	53.6	52.5	50.5	49.7
1	-	-	51.6	-	-	47.9
2	-	-	50.7	-	-	47.1
3	-	-	50.1	-	-	46.6
4	-	-	49.7	-	-	46.2
5	-	-	49.5	-	-	46.1
6	-	-	**49.3**	-	-	**46.0**

better after adaptation than the system with 16. The "#Iters CMLLR" refers to the number of E-M steps we do to obtain the CMLLR matrix in test time. In all cases the supervision hypothesis is the decoding output of the speaker independent system that was the input to the SAT training process. We do not rebuild the tree during SAT training. We use a regression tree with two classes: speech and silence. The top line (SAT epoch 0) refers to a speaker-independently trained system. Each epoch of SAT training starts with the model from the previous epoch, does three passes of E-M on the CMLLR matrices starting from the default value, and then does four iterations of E-M to update the model. The total improvement from SAT (comparing Epoch 0 to Epoch 6) is about 4.0% absolute.

Table 11 shows similar results for the SGMM system, with zero, one and two passes of CMLLR adaptation and with and without SAT. We do not use multiple regression classes, but we estimate the CMLLR transform only on speech (not silence) in test time; this was found to be important. In training time we adapt on both speech and silence as this appeared to be slightly better. The SAT training regime differs from the baseline HTK case in that we re-estimate CMLLR from scratch on each iteration (apart from the first eight, where we do not use CMLLR), but using only one iteration of E-M. In test time the setup is also different: we use the same model to do the first pass decoding and the final decoding (as we did not see a clear benefit from using a non-SAT model to do the first pass decoding), and in the case of two-iteration CMLLR we decode again before the second iteration of E-M.

With non-SAT models, the behavior of CMLLR is quite similar between

the SGMM and the HTK baseline: with a bigram LM, going from no CMLLR to multi-pass CMLLR, the HTK baseline improves from 55.8% to 53.6% (3.9% relative), and the SGMM system improves from 49.1% to 47.4% (3.5% relative). The big difference is in the effect of SAT: the HTK system (bigram decoding) improves from 53.6% to 49.3% (8.0% relative), while the SGMM system barely improves at all. However, the difference between the SGMM and baseline system is large enough that even with SAT, we are still doing better than the HTK system: from 49.3% to 47.4%, an improvement of 3.9% relative. Our best explanation for the different behavior with SAT is that our baseline system has no generic de-correlating transform such as HLDA [22] or MLLT [16], (or equivalently, global semi-tied covariance (STC) [9]), so SAT may be de-correlating the features. Our SGMM system is full covariance, so we would expect no improvement from this effect. See [36, Tables 8.18 and 8.21] for evidence that, in combination with HLDA, it is possible for SAT to give little or no improvement, either with diagonal or full-covariance-like (SPAM) models.

Table 11: SGMM with CMLLR – English %WER

		Bigram			Trigram		
Train	SAT	# Iters CMLLR					
Epoch	y/n	0	1	2	0	1	2
1	n	52.5	51.1	50.9	50.5	49.2	49.1
2	n	51.4	50.0	49.5	49.0	47.8	47.6
3	n	50.9	49.5	48.9	48.8	47.6	47.2
4	n	50.5	49.1	48.6	48.5	47.1	46.8
5	n	50.0	48.7	48.2	48.3	46.8	46.7
6	n	49.6	48.1	47.9	47.9	46.2	46.1
7	n	**49.1**	47.9	**47.4**	47.5	45.7	45.5
8	n	49.3	**47.8**	47.5	**47.5**	**45.7**	**45.5**
1	y	52.5	51.1	50.9	50.5	49.2	49.1
2	y	51.5	50.0	49.6	49.2	47.9	47.7
3	y	51.0	49.5	49.2	49.2	47.4	47.2
4	y	50.8	48.7	48.5	48.6	47.0	46.7
5	y	50.1	48.2	48.0	48.5	46.6	46.3
6	y	49.8	48.1	47.9	47.9	46.0	46.0
7	y	49.3	47.8	47.4	47.5	45.9	45.6
8	y	**49.0**	**47.7**	**47.4**	**47.4**	**45.6**	**45.3**

8.6. Adaptation with speaker vectors

We also investigated an adaptation approach which is specific to our model: the term $\mathbf{N}_i\mathbf{v}^{(s)}$ in the expression for the model means. This is quite different from CMLLR adaptation because the number of parameters to learn is much smaller: 39 in this case. Table 12 gives results with speaker vectors; note that this system was trained with speaker vectors so the first column without speaker-vectors in test time is not the true speaker independent baseline (refer to the first column of the upper part of Table 11). We used this decoding as supervision to train the speaker vectors in test time[8]. The use of the "speaker vectors" by themselves is not quite as effective as CMLLR: compare the best "speaker vector only" number of 47.8% in the third column of Table 12, with the best bigram CMLLR number from Table 11 which is 47.4%. Using 49.1% as the speaker independent bigram baseline, speaker vectors by themselves give 76% of the improvement of CMLLR. However, the speaker vectors have many fewer parameters so they can be estimated on less data; also, they are somewhat additive with CMLLR, with the best trigram-decoded number changing from 45.3% with CMLLR only in Table 11 to 44.3% in Table 12. The corresponding bigram WERs are 47.2% and 46.7%. Thus, adding speaker vectors to CMLLR gives us 2.2% and 1.1% relative improvement with trigram and bigram decoding respectively.

Table 12: SGMM with speaker vectors and/or CMLLR – English, %WER

#Iters	Bigram					Trigram	
Vecs:	0	1	2	1	1	1	1
CMLLR:	0	0	0	1	2	1	2
Epoch 1	52.5						
2	51.4						
3	51.8	49.9	48.9	48.4	48.1	46.5	46.1
4	51.4	48.6	48.5	47.7	47.5	45.6	45.3
5	51.0	48.6	48.2	47.3	47.1	45.3	45.0
6	50.6	48.2	47.9	47.0	46.8	45.0	45.0
7	50.2	**47.9**	**47.8**	**46.7**	46.5	44.6	44.4
8	**50.1**	48.0	47.9	48.1	**46.4**	**44.5**	**44.3**

[8]This is not optimal, but it is more convenient, and we saw no clear benefit from using a speaker-independently trained model in the first pass of decoding.

8.7. Basis representation of CMLLR

We also did experiments with the basis representation of the CMLLR transform given in Equation (12). This allows us to estimate a much smaller number of parameters for each speaker. These results are in Table 13. We decoded with and without CMLLR, and with and without speaker vectors; we adapted either per speaker or per segment, always with one iteration of E-M per parameter. The model used when decoding with speaker vectors is a separate model, trained with speaker vectors. The number of basis elements for the experiments with and without speaker vectors were each optimized to minimize the WER on the per-segment decoding experiments. What we find is that the basis essentially makes no difference to the results when the amount of adaptation data is sufficient (i.e. for per-speaker adaptation), but it improves the results dramatically when we are adapting on the per-segment level. Comparing the best results in each column, we see that when we do not use the basis, per-segment adaptation gives 0% or 21% as much improvement as per-speaker adaptation (with no speaker vectors, and with speaker vectors, respectively). If we use the basis, the corresponding numbers are 80% and 84%. Thus, the basis takes CMLLR adaptation from being almost completely ineffective to being very effective, when the adaptation is done per segment. The mean and median speaker lengths in our English test set are 164 and 165 seconds respectively, and the mean and median segment lengths are 1.8 and 2.2 seconds.

Table 13: SGMM, CMLLR with and without basis – English, bigram LM

Vecs:	No					Yes			
CMLLR:	No	Yes							
Basis:	-	No		$B=100$		No		$B=50$	
Adapt:	-	spk	seg	spk	seg	spk	seg	spk	seg
Epoch 1	52.5	50.6	52.4	50.7	51.3				
2	51.5	49.7	51.1	49.7	50.2				
3	50.9	49.4	50.7	49.3	49.9	47.9	50.5	48.0	48.9
4	50.5	48.7	50.3	48.7	49.5	47.5	49.9	47.5	48.6
5	50.0	48.3	49.8	48.4	48.9	47.1	49.3	47.2	48.3
6	49.6	48.0	49.5	48.0	48.6	47.0	48.9	47.0	47.6
7	**49.1**	**47.6**	**49.1**	**47.6**	48.0	**46.7**	**48.6**	**46.6**	47.3
8	49.3	47.6	49.2	47.6	**47.9**	46.7	48.6	46.6	**47.0**

8.8. Convergence in training

We now review the convergence behavior of the training procedures. Table 14 shows the amount of auxiliary function improvement obtained per frame from each parameter type on various iterations of training, along with the total auxiliary function improvement per frame[9]. The rightmost two columns display the likelihood, and the likelihood change from this iteration to the next. Generally with E-M updates, this likelihood change should never be less than the total auxiliary function change. This is not technically true here because we cannot prove convergence when combining certain pairs of update types on the same iteration (principally, any two of $\mathbf{M}_i$, $\mathbf{N}_i$ and $\mathbf{v}_{jm}$), but the condition seems to hold anyway. The only exceptions that can be seen in the table are after the first iteration of Epochs 3 and 8, where the likelihood decreases; this results from the sub-state splitting procedure described in Section 5.8. The negative auxiliary function improvement for the variances in Epoch 1, iteration 2 is due to a quite aggressive variance flooring constant $f = 0.2$ (c.f. Section 5.7); the larger than expected likelihood increase after Epoch 1, iteration 8 is due to the shift to "self-alignment".

Table 15 relates to the weight update, which is iterative within each update phase. The experiments we describe here were run with the "parallel" weight update of Algorithm 5.2, because the WER was slightly better (although this was not consistent). In this case the "convergence check" takes place on each iteration after updating all $\mathbf{w}_i$. We ran the weight update for $P_w = 3$ iterations during each update phase. In the training run, the convergence check failed (leading to halving the step size) on only two iterations: Epoch 6, iteration 2 and Epoch 8, iteration 2. In both cases this happened only on the first iteration of the iterative weight update, and we halved the step size only once. In Table 15(a) we display the auxiliary function improvements on each of the three iterations within selected update phases. On the left we show the change in the quadratic approximated auxiliary function of Equation (70), and on the right the change in the "real" auxiliary function of Equation (68). The crossed out numbers correspond to the auxiliary function changes before halving the step size. We also show, in Table 15(b), the corresponding figures for the sequential version of the update. The convergence is

[9]In the case of the weight update, Table 14 shows the change in the "real" auxiliary function of Equation (68) rather than the quadratic approximation of (70), although these are typically about the same. The training run displayed is from the system tested in the leftmost column of Table 13.

Table 14: Auxiliary function and likelihood changes

Iter	Auxiliary function change						Like	Δ Like
	$\mathbf{v}_{jm}$	$\mathbf{w}_i$	$\mathbf{M}_i$	$\mathbf{\Sigma}_i$	c_{jm}	Tot		
	Epoch 1							
1	0.44	-	-	-	-	0.44	-65.05	0.56
2	0.036	1.49	0.43	-0.468*	0	1.49	-64.49	1.62
3	0.231	0.12	0.13	0.133	0	0.62	-62.87	0.67
4	0.040	0.026	0.091	0.056	0	0.21	-62.20	0.23
5	0.028	0.0071	0.037	0.033	0	0.11	-61.97	0.12
6	0.015	0.0040	0.026	0.018	0	0.064	-61.85	0.070
7	0.012	0.0026	0.018	0.014	0	0.046	-61.78	0.050
8	0.009	0.0018	0.013	0.009	0	0.033	-61.74	0.226†
	Epoch 2							
1	0.014	0.0043	0.035	0.026	0	0.079	-61.51	0.099
2	0.017	0.0020	-	0.014	0	0.023	-61.41	0.028
3	0.0009	0.0011	0.018	0.007	0	0.027	-61.34	0.039
...								
	Epoch 3							
1	0.0002	0.0002	0.006	0.001	0	0.008	-61.27	-0.087‡
2	0.059	0.0045	-	0.014	0.001	0.078	-61.35	0.087
...								
	Epoch 8							
1	0.0033	0.0001	0.015	0.001	0.0001	0.019	-60.21	-0.025‡
...								

* Negative due to flooring. † Shift to self-alignment. ‡ Due to splitting.

similar to the parallel version; the "real" auxiliary function improvement (on the right) is almost the same as before. In the sequential update we also very rarely have to reduce the learning rate for any individual i; we only observed this happening on epoch 6, iteration 2 (we did not run this experiment past epoch 6), and only for five different values of i; the learning rate never had to be halved more than once. The conclusion is that these two approaches are both quite stable and effective.

Figure 3 shows the convergence of the CMLLR auxiliary function with iteration index p. Convergence is acceptably fast. Figure 4 displays likelihood plotted against training epoch, showing convergence of our overall training procedure. This run, which was the multilingual system of Table 7, used 16 iterations per epoch. For selected epochs, we display what happens if we do more iterations within the same epoch (rather than moving on to the

Table 15: Convergence of weight update

(a) "Parallel" version

Epoch /Iter	Quadratic auxf change				Real auxf change			
	Iteration of weight update							
	1	2	3	Total	1	2	3	Total
1/1	0.518	0.293	0.128	0.938	0.850	0.451	0.191	1.492
1/2	0.073	0.015	0.004	0.092	0.097	0.022	0.006	0.125
...								
6/2	~~0.0007~~				~~−0.0018~~			
	0.0005	0.0002	0.00002	0.0008	0.0005	0.0003	0.00003	0.0008

(b) "Sequential" version

Epoch /Iter	Quadratic auxf change				Real auxf change			
	Iteration of weight update							
	1	2	3	Total	1	2	3	Total
1/1	0.566	0.318	0.142	1.028	0.846	0.456	0.195	1.497
1/2	0.075	0.018	0.005	0.098	0.097	0.022	0.006	0.124

next epoch, with the attendant sub-state splitting). We omit the first few iterations of the first epoch because they would change the graph scale. Note that at the beginning of Epoch 2 we shift to using the SGMM for Viterbi alignment, and from Epoch 5 we begin to split sub-states.

9. Conclusions

We have introduced a new kind of acoustic model, the Subspace Gaussian Mixture Model (SGMM). This model is an example of a larger class of generative models where the parameters of the model are not the parameters of the distribution (e.g. the means and variances), but instead *generate* the parameters of the distribution. This makes it possible to describe quite complex distributions in a compact way. It will often be possible, as it is for this model, to evaluate and train these models quite efficiently. In our case we arrange the computation in such a way that each additional Gaussian evaluation is a dot product, and we make use of the structure of the model to prune the Gaussians we evaluate on each frame. A key advantage of this type of model is that it has a relatively small amount of parameters tied to the acoustic state, with many of the model's parameters being globally

Figure 3: Convergence of CMLLR auxiliary function

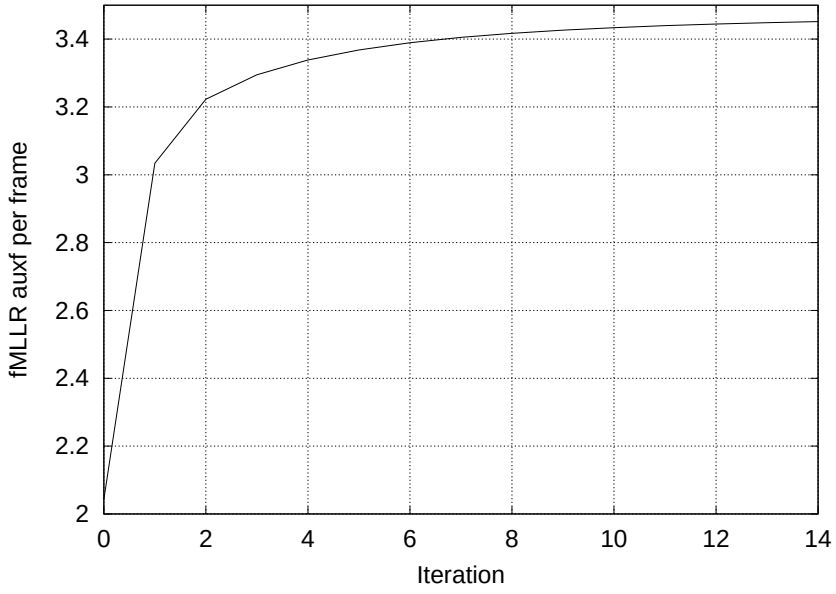
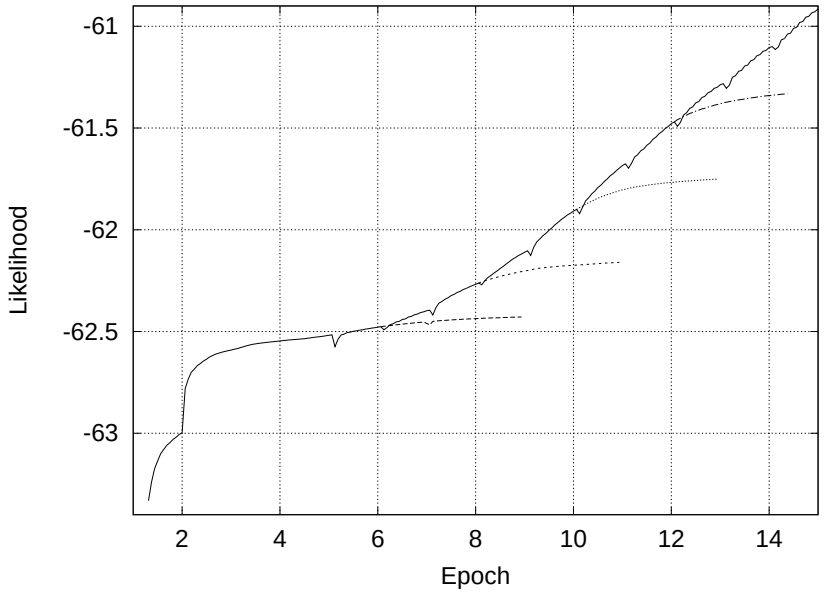


Figure 4: Convergence of overall procedure



shared. This makes it possible to train models on less in-domain data than would otherwise be necessary.

Our experiments on some of the CallHome databases, which each have about 15 hours of training data, confirmed that this model gives better results than a traditional GMM. These experiments were with Maximum Likelihood training, but experiments reported in [33] show that this approach can be combined with discriminative training. The amount of improvement varies depending on whether or not speaker adaptation is used. The improvement in the speaker independent condition is very large; for instance, on English, WER decreased from 52.3% to 47.5% (9.1% relative), but it would probably be less if the baseline system had HLDA. The improvement on the fully adapted English system was smaller: 46.0% to 44.3%, or 3.7% relative, mainly due to a larger than expected improvement from Speaker Adaptive Training on the baseline, which we believe may be performing the same function as HLDA. Previous experiments on a different setup have shown larger improvements than this [29][10]. Experiments with un-adapted systems show that we can expect a further improvement of approximately 5.5% relative from leveraging out-of-language data, which is not possible with a conventional system. We also demonstrated a very large improvement of 16% relative, without speaker adaptation, when the amount of training data is limited to one hour. Much of this came from the ability to naturally leverage out-of-language data. This is just one of many possible scenarios that are possible with this type of model, in which the shared parameters are trained on a different data-set than the state-specific parameters.

We are excited about this approach because this model appears to give better results than a conventional HMM-GMM, and also because it offers many opportunities for modeling improvements that are not possible in a conventional model. This type of model should also make it possible to greatly speed up acoustic likelihood computation without seriously degrading performance; this was not a focus of our work for this paper. We are interested in collaboration with other researchers to further develop this approach.

[10]This reference is to a book chapter which at the time of writing, has not yet been published. The improvement was from 24.3% to 19.6% (19.3% relative) on English Broadcast News, 50h training data, Maximum Likelihood, VTLN+CMLLR+MLLR adaptation.

Acknowledgments

This work was conducted at the Johns Hopkins University Summer Workshop which was supported by National Science Foundation Grant Number IIS-0833652, with supplemental funding from Google Research, DARPA's GALE program and the Johns Hopkins University Human Language Technology Center of Excellence. BUT researchers were partially supported by Czech Ministry of Trade and Commerce project no. FR-TI1/034, Grant Agency of Czech Republic project no. 102/08/0707, and Czech Ministry of Education project no. MSM0021630528. Arnab Ghoshal was partially supported by the European Community's Seventh Framework Programme under grant agreement number 213850 (SCALE). Thanks to CLSP staff and faculty, to Tomas Kasparek for system support, to Patrick Nguyen for introducing the participants, to Mark Gales for advice and HTK help, and to the anonymous reviewers for their comments and suggestions.

[1] Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., Mohri, M., 2007. OpenFst: a general and efficient weighted finite-state transducer library. In: Proc. CIAA.

[2] Amari, S. I., 1998. Natural gradient works efficiently in learning. Neural computation 10 (2), 251–276.

[3] Anastasakos, T., McDonough, J., Schwartz, R., Makhoul, J., 1996. A Compact Model for Speaker-Adaptive Training. In: Proc. ICSLP.

[4] Burget, L., Schwarz, P., Agarwal, M., Akyazi, P., Feng, K., Ghoshal, A., Glembek, O., Goel, N., Karafiát, M., Povey, D., Rastrow, A., Rose, R. C., Thomas, S., 2010. Multilingual Acoustic Modeling for Speech Recognition based on Subspace Gaussian Mixture Models. In: Proc. ICASSP.

[5] Canavan, A., Graff, D., Zipperlen, G., 1997. CALLHOME American English Speech. Linguistic Data Consortium.

[6] Canavan, A., Graff, D., Zipperlen, G., 1997. CALLHOME German Speech. Linguistic Data Consortium.

[7] Canavan, A., Zipperlen, G., 1996. CALLHOME Spanish Speech. Linguistic Data Consortium.

[8] Gales, M. J. F., 1997. Maximum Likelihood Linear Transformations for HMM-based Speech Recognition. Computer Speech and Language 12, 75–98.

[9] Gales, M. J. F., 1999. Semi-Tied Covariance Matrices For Hidden Markov Models. IEEE Transactions on Speech and Audio Processing 7, 272–281.

[10] Gales, M. J. F., 2001. Multiple-cluster adaptive training schemes. In: Proc. ICASSP.

[11] Gales, M. J. F. and Young, S. J., 2007. The application of hidden Markov models in speech recognition. Foundations and Trends in Signal Processing 1(3), 195–304.

[12] Ghoshal, A., Povey, D., Agarwal, M., Akyazi, P., Burget, L., Feng, K., Glembek, O., Goel, N., Karafiát, M., Rastrow, A., Rose, R. C., Schwarz, P., Thomas, S., 2010. A Novel Estimation of Feature-space MLLR for Full Covariance Models. In: Proc. ICASSP.

[13] Godfrey, J. J., et al., 1992. Switchboard: Telephone speech corpus for research and development. In: Proc. ICASSP.

[14] Goel, N., Thomas, S., Agarwal, M., Akyazi, P., Burget, L., Feng, K., Ghoshal, A., Glembek, O., Karafiát, M., Povey, D., Rastrow, A., Rose, R. C., Schwarz, P., 2010. Approaches to automatic lexicon learning with limited training examples. In: Proc. ICASSP.

[15] Golub, van Loan, 1983. Matrix computations, 3rd Edition. Johns Hopkins University Press.

[16] Gopinath, R. A., 1998. Maximum Likelihood Modeling With Gaussian Distributions For Classification. In: Proc. ICASSP. pp. 661–664.

[17] Graff, D., 2003. English Gigaword. Linguistic Data Consortium.

[18] Hermansky, H., 1990. Perceptual linear predictive (PLP) analysis of speech. Journal of the Acoustical Society of America 87, 1738–1752.

[19] Kenny, P., Ouellet, P., Dehak, N., Gupta, V., 2008. A study of Interspeaker Variability in Speaker Verification. IEEE Trans. on Audio, Speech and Language Processing 16 (5), 980–987.

[20] Kingsbury, P., et al., 1997. CALLHOME American English Lexicon (PRONLEX). Linguistic Data Consortium.

[21] Kuhn, R., Perronnin, F., Nguyen, P., Rigazzio, L., 2001. Very Fast Adaptation with a Compact Context-Dependent Eigenvoice Model. In: Proc. ICASSP.

[22] Kumar, N., Andreou, A. G., 1998. Heteroscedastic discriminant analysis and reduced rank HMMs for improved speech recognition. Speech Communication 26 (4), 283–297.

[23] Lin, H., Deng, L., Droppo, J., Yu, D., Acero, A., 2008. Learning methods in multilingual speech recognition. In: Proc. NIPS, Vancouver, Canada.

[24] Mohri, M., Pereira, F., Riley, M., 2002. Weighted finite-state transducers in speech recognition. Computer Speech and Language 20 (1), 69–88.

[25] Povey, D., 2004. Discriminative Training for Large Vocabulary Speech Recognition. Ph.D. thesis, Cambridge University.

[26] Povey, D., 2009. A Tutorial Introduction to Subspace Gaussian Mixture Models for Speech Recognition. Tech. Rep. MSR-TR-2009-111, Microsoft Research.

[27] Povey, D., 2009. Subspace Gaussian Mixture Models for Speech Recognition. Tech. Rep. MSR-TR-2009-64, Microsoft Research.

[28] Povey, D., Burget, L., Agarwal, M., Akyazi, P., Feng, K., Ghoshal, A., Glembek, O., Goel, N. K., Karafiát, M., Rastrow, A., Rose, R. C., Schwarz, P., Thomas, S., 2010. Subspace Gaussian Mixture Models for Speech Recognition. In: Proc. ICASSP.

[29] Povey, D., Chu, S. M., Pelecanos, J., 2010. Approaches to Speech Recognition based on Speaker Recognition Techniques. Book chapter in forthcoming book on GALE project.

[30] Povey, D., Chu, S. M., Varadarajan, B., 2008. Universal Background Model Based Speech Recognition. In: Proc. ICASSP.

[31] Povey, D., Saon, G., 2006. Feature and model space speaker adaptation with full covariance Gaussians. In: Proc. Interspeech/ICSLP.

[32] Reynolds, A. D., Quatieri, T. F., Dunn, R., 2000. Speaker verification using adapted Gaussian mixture models. Digital Signal Processing 10 (1-3), 19–41.

[33] Saon, G., Soltau, H., Chaudhari, U., Chu, S., Kingsbury, B., Kuo, H.-K., Mangu, L., Povey, D., 2010. The IBM 2008 GALE Arabic Speech Transcription System. In: Proc. ICASSP.

[34] Schultz, T., Waibel, A., 2001. Language-independent and language-adaptive acoustic modeling for speech recognition. Speech Communication 35 (1), 31–52.

[35] Sim, K. C., Gales, M. J. F., 2005. Adaptation of Precision Matrix Models on Large Vocabulary Continuous Speech Recognition. In: Proc. ICASSP.

[36] Sim, K. C., Gales, M. J. F., 2006. Structured Precision Matrix Modelling for Speech Recognition. Ph.D. thesis, Cambridge University.

[37] Stolcke, A., 2002. SRILM - An Extensible Language Modeling Toolkit. In: Proc. ICSLP.

[38] Visweswariah, K., Goel, V., Gopinath, R., 2002. Maximum Likelihood Training Of Bases For Rapid Adaptation. In: Proc. ICSLP.

[39] Young, S., Evermann, G., Gales, M. J. F., Hain, T., Kershaw, D., Liu, X., Moore, G., Odell, J. J., Ollason, D., Povey, D., Valtchev, V., Woodland, P., 2009. The HTK Book (for version 3.4). Cambridge University Engineering Department.

[40] Young, S., Odell, J. J., Woodland, P. C., 1994. Tree-Based State Tying for High Accuracy Acoustic Modelling. In: Proc. 1994 ARPA Human Language Technology Workshop. pp. 304–312.

[41] Zavaliagkos, G., Siu, M., Colthurst, T., Billa, J., 1998. Using Untranscribed Training Data to Improve Performance. In: Proc. ICSLP.

Appendix A. Robust auxiliary function optimization methods

There are two types of optimization problem that arise in the methods described above that can cause problems if implemented without regard to the possibility of poor numerical matrix rank. We describe our optimization approaches here; see [26, Appendix G] for more discussion. In these procedures we also calculate the auxiliary function changes for diagnostic purposes.

A.1. Vector auxiliary function

The function $\hat{\mathbf{v}} = \text{solve\_vec}(\mathbf{H}, \mathbf{g}, \mathbf{v}, K^{\max})$ optimizes the auxiliary function

$$\mathcal{Q}(\mathbf{v}) = \mathbf{v}\cdot\mathbf{g} - \tfrac{1}{2}\mathbf{v}^T\mathbf{H}\mathbf{v} \tag{A.1}$$

and returns the updated value $\hat{\mathbf{v}}$. It requires the old value $\mathbf{v}$ because it aims to leave $\mathbf{v}$ unchanged in the null-space of $\mathbf{H}$. It also computes the auxiliary function change. $\mathbf{H}$ is assumed to be positive semi-definite. If $\mathbf{H}$ is invertible it is simply a question of setting $\hat{\mathbf{v}} = \mathbf{H}^{-1}\mathbf{g}$, but in many cases $\mathbf{H}$ will be singular either mathematically or to machine precision. Our solution involves a singular value decomposition on $\mathbf{H}$ and effectively leaves $\mathbf{v}$ unchanged in the null-space of $\mathbf{H}$, although this operates in a soft manner via flooring the singular values of $\mathbf{H}$. The singular value decomposition also helps to avoid a loss of precision that would otherwise result when $\mathbf{H}$ has poor condition. See Algorithm A.1.

Algorithm A.1 $\hat{\mathbf{v}} = \text{solve\_vec}(\mathbf{H}, \mathbf{g}, \mathbf{v}, K^{\max})$

Require: $\mathbf{H}$ symmetric positive semi-definite

if $\mathbf{H}$ is the zero matrix **then**
  $\hat{\mathbf{v}} \leftarrow \mathbf{v}$
  $\Delta\mathcal{Q} \leftarrow 0$
else
  $\bar{\mathbf{g}} \leftarrow \mathbf{g} - \mathbf{H}\mathbf{v}$
  Do the SVD $\mathbf{H} = \mathbf{U}\mathbf{L}\mathbf{V}^T$ (discard $\mathbf{V}$)
  $f \leftarrow \frac{\max_i l_{ii}}{K^{\max}}$ where l_{ii} are the diagonal elements of $\mathbf{L}$
  Compute the floored diagonal matrix $\tilde{\mathbf{L}}$ with $\tilde{l}_{ii} = \max(f, l_{ii})$
  $\hat{\mathbf{v}} \leftarrow \mathbf{v} + \mathbf{U}(\tilde{\mathbf{L}}^{-1}(\mathbf{U}^T\bar{\mathbf{g}}))$ (parentheses important)
  $\Delta\mathcal{Q} \leftarrow \mathbf{g}\cdot(\hat{\mathbf{v}}-\mathbf{v}) - \frac{1}{2}\hat{\mathbf{v}}^T\mathbf{H}\hat{\mathbf{v}} + \frac{1}{2}\mathbf{v}^T\mathbf{H}\mathbf{v}$
end if

A.2. Matrix auxiliary function

The matrix version of the above is a function which maximizes the auxiliary function

$$\mathcal{Q}(\mathbf{M}) = \mathrm{tr}\,(\mathbf{M}^T\mathbf{P}\mathbf{Y}) - \tfrac{1}{2}\mathrm{tr}\,(\mathbf{P}\mathbf{M}\mathbf{Q}\mathbf{M}^T) \tag{A.2}$$

where the solution if $\mathbf{Q}$ is invertible would be $\hat{\mathbf{M}} = \mathbf{Y}\mathbf{Q}^{-1}$. See Algorithm A.2.

Algorithm A.2 $\hat{\mathbf{M}} = \mathrm{solve\_mat}(\mathbf{Q}, \mathbf{Y}, \mathbf{P}, \mathbf{M}, K^{\max})$

Require: $\mathbf{P}$ and $\mathbf{Q}$ symmetric positive semi-definite

if $\mathbf{Q}$ is the zero matrix **then**
  $\hat{\mathbf{M}} \leftarrow \mathbf{M}$
  $\Delta\mathcal{Q} \leftarrow 0$
else
  $\bar{\mathbf{Y}} \leftarrow \mathbf{Y} - \mathbf{M}\mathbf{Q}$
  Do the SVD $\mathbf{Q} = \mathbf{U}\mathbf{L}\mathbf{V}^T$ (discard $\mathbf{V}$)
  $f \leftarrow \frac{\max_i l_{ii}}{K^{\max}}$ where l_{ii} are the diagonal elements of $\mathbf{L}$
  Compute the floored diagonal matrix $\tilde{\mathbf{L}}$ with $\tilde{l}_{ii} = \max(f, l_{ii})$
  $\hat{\mathbf{M}} \leftarrow \mathbf{M} + ((\bar{\mathbf{Y}}\mathbf{U})\tilde{\mathbf{L}}^{-1})\mathbf{U}^T$ (parentheses important)
  $\Delta\mathcal{Q} \leftarrow \mathrm{tr}\,((\hat{\mathbf{M}}-\mathbf{M})^T\mathbf{P}\mathbf{Y}) - \frac{1}{2}\mathrm{tr}\,(\mathbf{P}\hat{\mathbf{M}}\mathbf{Q}\hat{\mathbf{M}}^T) + \frac{1}{2}\mathrm{tr}\,(\mathbf{P}\mathbf{M}\mathbf{Q}\mathbf{M}^T)$
end if

Appendix B. Constrained MLLR update

B.1. Overview

Our update for CMLLR is based on a co-ordinate change in the parameters of $\mathbf{W}^{(s)} = \left[\mathbf{A}^{(s)};\ \mathbf{b}^{(s)}\right]$. Let us call the auxiliary function $\mathcal{Q}$. Viewing the elements of $\mathbf{W}^{(s)}$ as a vector, we do a co-ordinate change in this vector space such that the expected value of the matrix of second derivatives of $\mathcal{Q}$ with respect to this vector, is approximately proportional to the unit matrix. In this pre-transformed space we repeatedly find the optimal step size in the direction of the gradient, and because of the co-ordinate change this converges quite rapidly. This is a version of natural gradient descent [2]. The Hessian in the transformed space will not be an exact multiple of the unit matrix (because of the words "expected" and "approximately" above), but it will be close enough to make the gradient based method converge fast. This formulation is efficient for our style of model, more so than the obvious extension

of the standard approach [8] to full covariance models [35, 31]. Our style of update also makes it easier to estimate and apply the basis representation of Equation (12). We will not provide any derivations (see [26]), but we will try to give some idea of the purpose of each step of the process.

B.2. CMLLR: First pre-computation phase

The first phase of pre-computation is to compute the matrices $\mathbf{W}_{\text{pre}} = [\mathbf{A}_{\text{pre}}\ \mathbf{b}_{\text{pre}}]$ which normalizes the space in an LDA-like sense, plus its inverse transform $\mathbf{W}_{\text{inv}} = [\mathbf{A}_{\text{inv}}\ \mathbf{b}_{\text{inv}}]$ and an associated diagonal matrix $\mathbf{D}$ which corresponds to the eigenvalues in the LDA computation. These matrices are not speaker specific. This should be done after at least one or two iterations of speaker-independent training of the SGMM system. We only require the model and the per-state counts $\gamma_j = \sum_{m,i} \gamma_{jmi}$. We compute:

$$p(j) = \gamma_j / \sum_{j=1}^{J} \gamma_j \tag{B.1}$$

$$\mathbf{\Sigma}_W = \sum_{j,m,i} p(j) c_{jm} w_{jmi} \mathbf{\Sigma}_i \tag{B.2}$$

$$\boldsymbol{\mu}_{\text{avg}} = \sum_{j,m,i} p(j) c_{jm} w_{jmi} \boldsymbol{\mu}_{jmi} \tag{B.3}$$

$$\mathbf{\Sigma}_B = \left(\sum_{j,m,i} p(j) c_{jm} w_{jmi} \boldsymbol{\mu}_{jmi} \boldsymbol{\mu}_{jmi}^T \right) - \boldsymbol{\mu}_{\text{avg}} \boldsymbol{\mu}_{\text{avg}}^T. \tag{B.4}$$

After the above, we do the Cholesky decomposition $\mathbf{\Sigma}_W = \mathbf{L}\mathbf{L}^T$, compute $\mathbf{B} = \mathbf{L}^{-1} \mathbf{\Sigma}_B \mathbf{L}^{-T}$, do the singular value decomposition

$$\mathbf{B} = \mathbf{U}\mathbf{D}\mathbf{V}^T \tag{B.5}$$

keeping $\mathbf{D}$ for later use, and compute:

$$\mathbf{A}_{\text{pre}} = \mathbf{U}^T \mathbf{L}^{-1} \tag{B.6}$$

$$\mathbf{b}_{\text{pre}} = -\mathbf{A}_{\text{pre}} \boldsymbol{\mu}_{\text{avg}} \tag{B.7}$$

$$\mathbf{A}_{\text{inv}} = \mathbf{A}_{\text{pre}}^{-1} = \mathbf{L}\mathbf{U} \tag{B.8}$$

$$\mathbf{b}_{\text{inv}} = \boldsymbol{\mu}_{\text{avg}} \tag{B.9}$$

B.3. CMLLR: Second pre-computation phase

The second pre-computation phase relates to the use of parameter subspaces with CMLLR, i.e. the basis representation of the transformation matrix of Equation (12). This is not a necessary part of the overall scheme but may improve results if the amount of data to be adapted on is very small (e.g., less than 20 seconds). We compute the $D(D+1) \times D(D+1)$ matrix $\mathbf{M}$ which is the scatter of the vectorized form of a gradient quantity $\tilde{\mathbf{P}}$ that is computed within the CMLLR computation (see below):

$$\mathbf{M} = \sum_{s \in \mathcal{S}} \tfrac{1}{\beta^{(s)}} \mathrm{vec}(\tilde{\mathbf{P}}^{(s)}) \mathrm{vec}(\tilde{\mathbf{P}}^{(s)})^T \tag{B.10}$$

where $\mathcal{S}$ is the set of all speakers and $\mathrm{vec}(\mathbf{A})$ means concatenating the rows of $\mathbf{A}$. From $\mathbf{M}$ we compute the basis matrices $\tilde{\mathbf{W}}_b$ for $1 \leq b \leq B$ for some selected basis size (e.g. $B = 200$, c.f. [38]) by doing the sorted singular value decomposition $\mathbf{M} = \mathbf{U}\mathbf{L}\mathbf{V}^T$ and taking $\tilde{\mathbf{W}}_b$ as the b'th column of $\mathbf{U}$, turned back into a matrix (i.e. $\mathbf{u}_b = \mathrm{vec}(\tilde{\mathbf{W}}_b)$). The matrices $\tilde{\mathbf{W}}_b$ are not the same as the $\mathbf{W}_b$ of (12) because they are represented in the normalized co-ordinate space.

B.4. CMLLR: Update phase

The count, linear and quadratic statistics $\beta^{(s)}$, $\mathbf{K}^{(s)}$ and $\mathbf{S}_i^{(s)}$ needed for the CMLLR update are accumulated in (51) to (53). We start with a default value $\mathbf{W}^{(s)} = [\mathbf{I}\,;\,\mathbf{0}]$, or with an existing value of $\mathbf{W}^{(s)}$ if more than one E-M iteration is being done for this speaker. We will drop the speaker superscript below. The update phase is iterative. The outer-loop iteration index is $p = 1 \ldots P_{\mathrm{cmllr}}$, with P_{cmllr} typically in the range 5 to 20. On each iteration p, we first compute the quantity:

$$\mathbf{S} = \sum_i \mathbf{\Sigma}_i^{-1} \mathbf{W} \mathbf{S}_i. \tag{B.11}$$

We then measure the auxiliary function:

$$\mathcal{Q}(\mathbf{W}) \;=\; \beta \log |\det \mathbf{A}| + \mathrm{tr}\,(\mathbf{W}\mathbf{K}^T) - \tfrac{1}{2}\mathrm{tr}\,(\mathbf{W}^T\mathbf{S}) \tag{B.12}$$

where $\mathbf{A}$ is the first D columns of $\mathbf{W}$. The auxiliary function should never decrease from one iteration to the next. We compute the gradient of the

auxiliary function[11]

$$\mathbf{P} = \beta \left[\mathbf{A}^{-T} ; \mathbf{0} \right] + \mathbf{K} - \mathbf{S}. \tag{B.13}$$

The first step of co-ordinate transformation is:

$$\mathbf{P}' = \mathbf{A}_{\text{inv}}^{T} \mathbf{P} \mathbf{W}_{\text{pre}}^{+\ T} \tag{B.14}$$

where we use the notation $\cdot^+$ to represent appending a row $[0\ 0\ \ldots 0\ 1]$ to a matrix. $\mathbf{P}'$ is the gradient in a feature-normalized space. The next step transforms $\mathbf{P}'$ to $\tilde{\mathbf{P}}$; this step is equivalent to multiplying by the inverse Cholesky factor (i.e. inverse square root) of the expected per-frame Hessian in the feature-normalized space. Think of this expected Hessian as a $D(D+1) \times D(D+1)$ matrix with a sparse structure. Below, we refer to elements d_i of the diagonal matrix of eigenvalues $\mathbf{D}$ of (B.5). For $1 \leq i \leq D$ and for $1 \leq j < i$, we do:

$$\begin{aligned} \tilde{p}_{ij} &= (1 + d_j)^{-\frac{1}{2}} p'_{ij} && \text{(B.15)} \\ \tilde{p}_{ji} &= -\left(1 + d_i - (1 + d_j)^{-1}\right)^{-\frac{1}{2}} (1 + d_j)^{-1} p'_{ij} \\ &\quad + \left(1 + d_i - (1 + d_j)^{-1}\right)^{-\frac{1}{2}} p'_{ji} && \text{(B.16)} \end{aligned}$$

and for the diagonal and the offset term, for $1 \leq i \leq D$,

$$\begin{aligned} \tilde{p}_{ii} &= (2 + d_i)^{-\frac{1}{2}} p'_{ii} && \text{(B.17)} \\ \tilde{p}_{i,(D+1)} &= p'_{i,(D+1)} && \text{(B.18)} \end{aligned}$$

If we were doing this for training the basis, we would stop at this point, save β and $\tilde{\mathbf{P}}$ or do the accumulation of (B.10), and terminate the loop over p. Otherwise, the next step is to compute the proposed parameter change $\tilde{\mathbf{\Delta}}$ in the completely transformed space. If we are not using the basis, we compute:

$$\tilde{\mathbf{\Delta}} = (1/\beta)\tilde{\mathbf{P}}. \tag{B.19}$$

If using the basis, this becomes:

$$\tilde{\mathbf{\Delta}} = \frac{1}{\beta} \sum_{b=1}^{B} \tilde{\mathbf{W}}_b \text{tr}\left(\tilde{\mathbf{P}}^T \tilde{\mathbf{W}}_b\right). \tag{B.20}$$

[11] From differentiating (B.12) it may seem that we are missing a factor of $\frac{1}{2}$, but remember that $\mathbf{S}$ is a function of $\mathbf{W}$.

For simplicity and compatibility with the "no-basis" version of CMLLR, we never explicitly store the coefficients $a_b^{(s)}$ of Equation (12). Equation (B.20) is the same as (B.19) but restricted to the subspace defined by the basis. We remark that the decision of whether to use a basis and if so, what basis size B to use, can be made per speaker. The basis matrices $\tilde{\mathbf{W}}_b$ correspond to eigenvectors of a matrix, so if we just store them in decreasing order of eigenvalue we can truncate the list after we decide the basis size B.

The next step in the algorithm is the reverse co-ordinate transformation from $\tilde{\mathbf{\Delta}}$ to $\mathbf{\Delta}'$. We write the elements of the matrix $\mathbf{\Delta}$ as δ_{ij}. For $1 \leq i \leq D$ and for $1 \leq j < i$,

$$\delta'_{ij} = (1+d_j)^{-\frac{1}{2}} \tilde{\delta}_{ij} - \left(1 + d_i - (1+d_j)^{-1}\right)^{-\frac{1}{2}} (1+d_j)^{-1} \tilde{\delta}_{ji} \quad \text{(B.21)}$$

$$\delta'_{ji} = \left(1 + d_i - (1+d_j)^{-1}\right)^{-\frac{1}{2}} \tilde{\delta}_{ji} \quad \text{(B.22)}$$

and for the diagonal and the offset terms, for $1 \leq i \leq D$,

$$\delta'_{ii} = (2+d_i)^{-\frac{1}{2}} \tilde{\delta}_{ii} \quad \text{(B.23)}$$

$$\delta'_{i,(D+1)} = \tilde{\delta}_{i,(D+1)}. \quad \text{(B.24)}$$

We then transform $\mathbf{\Delta}'$ to $\mathbf{\Delta}$ with:

$$\mathbf{\Delta} = \mathbf{A}_{\text{inv}} \mathbf{\Delta}' \mathbf{W}_{\text{pre}}^{+}. \quad \text{(B.25)}$$

At this point, it is advisable to check that the following two equalities hold as this will help to detect implementation errors in the co-ordinate transformations:

$$\text{tr}\,(\mathbf{\Delta}\mathbf{P}^T) = \text{tr}\,(\mathbf{\Delta}'\mathbf{P}'^T) = \text{tr}\,(\tilde{\mathbf{\Delta}}\tilde{\mathbf{P}}^T). \quad \text{(B.26)}$$

We will compute the optimal step size k in the direction $\mathbf{\Delta}$, i.e. $\mathbf{W} \leftarrow \mathbf{W} + k\mathbf{\Delta}$, so the next phase is to iteratively compute k. Its final value should typically be close to 1. We are maximizing the auxiliary function:

$$\mathcal{Q}(k) = \beta \log\det(\mathbf{A} + k\mathbf{\Delta}_{1:D,1:D}) + k\,m - \tfrac{1}{2}k^2 n \quad \text{(B.27)}$$

$$m = \text{tr}\,(\mathbf{\Delta}\mathbf{K}^T) - \text{tr}\,(\mathbf{\Delta}\mathbf{S}^T) \quad \text{(B.28)}$$

$$n = \sum_i \text{tr}\,\left(\mathbf{\Delta}^T \mathbf{\Sigma}_i^{-1} \mathbf{\Delta} \mathbf{S}_i\right) \quad \text{(B.29)}$$

where $\mathbf{\Delta}_{1:D,1:D}$ is the first D columns of $\mathbf{\Delta}$ and $\mathbf{A}$ is the first D columns of $\mathbf{W}$. We use an iterative Newton's method optimization for k, starting from

$k = 0$. A small number of iterations, e.g. $R = 5$, should be sufficient. On each iteration r, we compute

$$\mathbf{B} = (\mathbf{A} + k\boldsymbol{\Delta}_{1:D,1:D})^{-1}\boldsymbol{\Delta}_{1:D,1:D} \tag{B.30}$$

$$d_1 = \beta \mathrm{tr}\,(\mathbf{B}) + m - kn \tag{B.31}$$

$$d_2 = -\beta(\mathrm{tr}\,\mathbf{BB}) - n \tag{B.32}$$

where d_1 and d_2 are the first and second derivatives of (B.27) with respect to k, and then generate a new candidate value of k:

$$\hat{k} = k - (d_1/d_2). \tag{B.33}$$

At this point we should check that the value of (B.27) does not decrease when replacing k with $\hat{k}$, and if it does we should keep halving the step size $\hat{k} \leftarrow (k + \hat{k})/2$ and re-trying. There should be a limit in how many times this halving is done, to prevent an infinite loop; close to convergence, numerical roundoff can cause spurious decreases. When we have ensured that the auxiliary function does not decrease we would set $k \leftarrow \hat{k}$ and continue the loop over r. After completing R iterations, we do the update:

$$\mathbf{W} \leftarrow \mathbf{W} + k\boldsymbol{\Delta}. \tag{B.34}$$

and continue the outer loop over p. The auxiliary function of Equation (B.12) should increase or not change on each iteration. The amount of increase in (B.12) from iteration p to $p+1$ should be the same as the increase in (B.27) from $k = 0$ to the final k value computed on iteration p.