A TUTORIAL-STYLE INTRODUCTION TO SUBSPACE GAUSSIAN MIXTURE MODELS FOR SPEECH RECOGNITION

Daniel Povey[*]

Microsoft,
One Microsoft Way, Redmond, WA 98052
dpovey@microsoft.com

ABSTRACT

This is an in-depth, tutorial-style introduction to the techniques involved in training a factor analyzed style of speech recognition system. Algorithms are explained in detail, with an emphasis on the how-to rather than the derivations. The recipe described here is both an extension to and a special case of the prior work we have done. Changes include a simplification of the procedure used to initialize these models, the introduction of "sub-models" which saves memory and may have modeling advantages, an extended approach to factor based speaker adaptation that uses the sub-models, and a mechanism to estimate a subspace-constrained version of Constrained MLLR transforms in this framework.

***Index Terms*—** Speech Recognition, Universal Background Model, Factor Analysis

1. ABOUT THIS DOCUMENT

This document was originally created by Dan Povey in the months leading up to the Johns Hopkins summer workshop on speech processing, for the research group named "Low Development Cost, High Quality Speech Recognition for New Languages and Domains". It is intended to serve as a tutorial-style introduction to the use of subspace GMMs for speech recognition. It has been through various extensions and revisions, and even during the workshop various parts of it have been changed in order to fix errors discovered by other workshop participants and modify update formulas and the corresponding derivations where problems were discovered. Also some parts of the document have been changed in order to be compatible with the way we have actually implemented these techniques (for instance, not using "offset terms" which simplifies the mathematics). Particular thanks go to Lukas Burget and Arnab Ghoshal, who found and helped to fix a number of problems.

2. INTRODUCTION

Section 3 describes some nonstandard notation that we will use throughout the rest of the document. The model is introduced in Section 4, where we describe first the basic idea and then the various extensions which we intend to build on top of it. Section 5 describes the relationship between this method and previous work.

Section 6 describes in general terms the process of training this type of model and discusses the kind of code architecture we envisage to support it. Section 7 discusses the initialization of the Gaussian Mixture Model (GMM) that is used to initialize the main model and for pruning. Section 8 describes the initialization of the actual model. Section 9 describes the process used for fast likelihood evaluation given the main model (this is needed in training and testing). Section 10 describes the various statistics accumulation processes required. Section 11 describes the various kinds of model update. The derivation and equations for the constrained MLLR estimation approach we are using is in Sections 12 and 13; Section 14 summarizes the procedures used when applying these to the model we are using here.

Appendix A describes an algorithm for fast computation of the top eigenvectors of a scatter matrix, which is useful in the estimation of parameter subspaces for constrained MLLR. Appendix B contains a proof related to singular value decomposition which we use in other parts of the document. Appendix C describes a technique for using prior probabilities with the model we describe, which we have moved to an appendix to avoid cluttering the main document. Appendix D describes a probability model for offsets from the mean of a distribution, which is used in our model as part of a prior distribution. Appendix E describes an algorithm for Maximum Likelihood clustering of a number of Gaussians into clusters, each represented by a Gaussian. Appendix F contains derivations of some formulas used in the main text. Appendix G describes procedures for maximizing auxiliary functions involving matrices of reduced rank. Appendix H describes a process for limiting the condition of a matrix. Appendix I describes a more general process for flooring symmetric matrices. Appendix J describes how to estimate and use a prior distribution over the projection matrices estimated in this scheme. Appendix K describes a method for renormalizing the phonetic

3. NOTATION

We use some non-standard notation to simplify the equations. This is summarized here:

$\mathbf{v}^+$	Vector $\mathbf{v}$ extended with a 1, i.e. $\begin{bmatrix} v_1 \\ \vdots \\ v_n \\ 1 \end{bmatrix}$.
$\mathbf{v}^-$	Vector $\mathbf{v}$ with its last element removed
$\mathbf{M}^+$	Matrix $\mathbf{M}$ with an extra last row equal to $[0\ 0\ \ldots\ 0\ 1]$
$\mathbf{M}^{+0}$	Matrix $\mathbf{M}$ with an extra zero row appended
$\mathbf{M}^-$	Matrix $\mathbf{M}$ with its last row removed
$\mathbf{M}^{--}$	Matrix $\mathbf{M}$ with its last row and column removed
$\mathbf{M}^{-C}$	Matrix $\mathbf{M}$ with its last column removed

[*]Thanks to Lukas Burget, Arnab Ghoshal, Rick Rose and Geoff Zweig for comments and suggestions.

4. SUBSPACE GAUSSIAN MIXTURE MODEL

In this section we describe the modeling approach we are using. Rather than immediately write down the model, we build up in complexity starting from the basic idea. This should enable the reader to distinguish between the core ideas and the features that were built on top. Section 4.1 describes the basic model, which is based on a subspace representation of the mean parameters of a shared GMM structure. Section 4.4 describes the addition of sub-states, which involves having a mixture within each acoustic state of the basic model. Section 4.5 describes the addition of "speaker factors", which makes the model mean a sum of two mirror-image terms, one coming from the acoustic state and one from the speaker. Section 4.6 describes the introduction of "sub-models", in which the model described above is split up into a number of different models, each applying to a particular general region of acoustic space, and likelihoods are obtained by summing over the sub-models.

4.1. Basic model

In this section we describe the most basic form of the model, without speaker adaptation or sub-states. We use the index $1 \leq i \leq I$ for the Gaussians in the shared GMM (e.g. $I = 750$), and the index $1 \leq j \leq J$ for the clustered phonetic states (e.g. $J = 8000$ for a reasonably typical large vocabulary system). Let the feature dimension be $1 \leq d \leq D$, e.g. $D = 40$, and let the subspace dimension be $1 \leq s \leq S$, e.g. $S = 50$, with $S \leq ID$. The "subspace" of dimension S is a subspace of the total parameter space of the means of the GMM, which is of size ID.

For each state j, the probability model $p(\mathbf{x}|j)$ is:

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

$$\boldsymbol{\mu}_{ji} = \mathbf{M}_i \mathbf{v}_j^+ \quad (2)$$

$$w_{ji} = \frac{\exp(\mathbf{w}_i^T \mathbf{v}_j^+)}{\sum_{i'=1}^{I} \exp(\mathbf{w}_{i'}{}^T \mathbf{v}_j^+)} \quad (3)$$

Thus, each state has a shared number of mixtures (e.g., $I = 750$). The means vary linearly with the state-specific vector $\mathbf{v}_j$. We use $\mathbf{v}_j^+$ to represent the same vector extended with a 1, to handle constant offsets. The log weights prior to normalization also vary linearly with $\mathbf{v}_j$. The parameters of the system are the mean-projection matrices $\mathbf{M}_i \in \Re^{D \times (S+1)}$, the weight-projection vectors $\mathbf{w}_i \in \Re^{(S+1)}$, the variances $\boldsymbol{\Sigma}_i \in \Re^{D \times D}$, and the state-specific vectors $\mathbf{v}_j \in \Re^S$.

4.2. Discussion of model size

To give the reader a feel for the number of parameters involved, for the values of I, J, D and S mentioned above the total number of parameters would be, from most to fewest parameters: mean-projections, $ID(S+1) = 750 \times 40 \times (50+1) = 1.53 \times 10^6$; variances, $\frac{1}{2}ID(D+1) = \frac{750 \times 40 \times 41}{2} = 0.615 \times 10^6$; state-specific vectors, $JS = 0.4 \times 10^6$; weight-projections, $IS = 750 \times (50+1) = 38.25 \times 10^3$. Thus the total number of parameters is 2.58×10^6, and most of the parameters are globally shared, not state-specific. For reference, a typical mixture-of-Gaussians system optimized on a similar amount of training data might have 100000 Gaussians in total, each with a 40-dimensional mean and variance, which gives us 8×10^6 parameters total, more than twice the subspace GMM system. Note that the quantity of state-specific parameters in the subspace GMM system is less than one tenth of that in the normal GMM system. For this reason, we extend the model to include mixtures of sub-states.

4.3. Omitting the offset terms

The use of the "offset terms" implied by $\mathbf{v}^+$ is optional; the only advantage of constraining the last element of $\mathbf{v}^+$ to be 1 is that we do not need to estimate that parameter, and by doing away with that constraint we only have to estimate one more parameter out of, say, 50. This simplifies the equations. Without the offset, we would have:

$$\boldsymbol{\mu}_{ji} = \mathbf{M}_i \mathbf{v}_j \quad (4)$$

$$w_{ji} = \frac{\exp(\mathbf{w}_i^T \mathbf{v}_j)}{\sum_{i'=1}^{I} \exp(\mathbf{w}_{i'}{}^T \mathbf{v}_j)}, \quad (5)$$

$$(6)$$

and this would affect the dimensions of $\mathbf{M}_i$ which would now have dimension $D \times S$, and of $\mathbf{w}_i$ which would now have dimension S. In the following, we will show the equations *with* the offset terms but will make clear which terms and operators would disappear if we were not using the offsets, by putting all offset terms and operators in red (this may render as light gray when printed in black and white). For example, we would have:

$$\boldsymbol{\mu}_{ji} = \mathbf{M}_i \mathbf{v}_j^+, \quad (7)$$

with $\mathbf{M}_i \in \Re^{D \times (S+1)}$, so removing everything in red would give us the form of the equations without the offsets.

4.4. Subspace mixture model with sub-states

The subspace mixture model with sub-states is the same as in Equations (1) to (3) except each state is now like a mixture of states; each state j has sub-states numbered $1 \leq m \leq M_j$ with associated vectors $\mathbf{v}_{jm}$ and mixture weights c_{jm} with $\sum_{m=1}^{M_j} c_{jm} = 1$; we can write out the model as:

$$p(\mathbf{x}|j) = \sum_{m=1}^{M_j} c_{jm} \sum_{i=1}^{I} w_{jmi} \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_{jmi}, \boldsymbol{\Sigma}_i) \quad (8)$$

$$\boldsymbol{\mu}_{jmi} = \mathbf{M}_i \mathbf{v}_{jm}^+ \quad (9)$$

$$w_{jmi} = \frac{\exp(\mathbf{w}_i^T \mathbf{v}_{jm}^+)}{\sum_{i'=1}^{I} \exp(\mathbf{w}_{i'}^T \mathbf{v}_{jm}^+)}. \quad (10)$$

It is useful to think about the sub-states as corresponding to Gaussians in a mixture of Gaussians, and as we describe later, we use a variant of a familiar Gaussian mixing-up procedure to increase the number of states. This model is in effect a mixture of mixtures of Gaussians, with the total number of Gaussians in each state being equal to I J_m. Clearly this large size could be expected to lead to efficiency problems. Later we will show how despite this, likelihoods given this model can be computed in a time similar to a normal diagonal mixture of Gaussians.

4.5. Subspace mixture model with speaker vectors

Another useful extension to the basic subspace GMM framework is a technique that uses speaker vectors, where each speaker s will be described by a speaker vector $\mathbf{v}^{(s)}$ of dimension T (e.g. we might

use $T = 50$, the same as the subspace dimension S). The projected mean now becomes:

$$\boldsymbol{\mu}_{jmi}^{(s)} = \mathbf{M}_i \mathbf{v}_{jm}^{+} + \mathbf{N}_i \mathbf{v}^{(s)+}, \quad (11)$$

so $\mathbf{N}_i \mathbf{v}^{(s)+}$ becomes a speaker-specific offset to means $\boldsymbol{\mu}_{jmi}$ for speaker s. We extend the speaker vector $\mathbf{v}^{(s)}$ with a 1 (assuming we are using offsets) to $\mathbf{v}^{(s)+}$ for symmetry and to introduce the potential for code sharing, but the offset term (if used) is redundant with the one in $\mathbf{v}_{jm}^{+}$. In previous work [1, 2], we omitted the offset on the speaker vector. We do not make the mixture weights dependent on the speaker factor: this is for efficiency reasons, as it enables the speaker adaptation to be implemented as a feature-space offset for each Gaussian index i. The use of separate subspaces for each speech state and speaker is analogous to the "factor analysis" approach used in speaker identification [3]. Because the number of parameters to be estimated per speaker is so small, in practice we have actually been estimating these vectors for each speech segment of each speaker.

4.6. Sub-models

A further extension that we can try is a mixture of Subspace GMMs, which we call a mixture of "sub-models". Unlike the modifications described above, this has not been implemented before. This would involve initially partitioning the I Gaussians in the background GMM into clusters $1 \leq k \leq K$, each of size I_k. Then, we essentially duplicate the model above for each cluster k and sum over the clusters. In place of Equations (8) to (10) (but using the speaker-factor modification of Equation (11)), we now have:

$$p(\mathbf{x}|j,s) = \sum_{k=1}^{k} \sum_{m=1}^{M_{jk}} c_{jkm} \sum_{i=1}^{I_k} w_{jkmi} \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_{jkmi}^{(s)}, \boldsymbol{\Sigma}_{ki}) \quad (12)$$

$$\boldsymbol{\mu}_{jkmi}^{(s)} = \mathbf{M}_{ki} \mathbf{v}_{jkm}^{+} + \mathbf{N}_{ki} \mathbf{v}_{k}^{(s)+} \quad (13)$$

$$w_{jkmi} = \frac{\exp(\mathbf{w}_{ki}^T \mathbf{v}_{jkm}^{+})}{\sum_{i'=1}^{I_k} \exp(\mathbf{w}_{ki'}^T \mathbf{v}_{jkm}^{+})}, \quad (14)$$

where the constraint on the mixture weights is now that $\sum_{k=1}^{k} \sum_{m=1}^{M_{jk}} c_{jkm} = 1$. Now we have K speaker vectors instead of just one. The main advantage of the use of sub-models is that on the one hand it reduces the memory required to train and decode, and on the other hand it allows us to build larger models for the same amount of memory. It also gives us a choice between making the speaker vectors $\mathbf{v}_k^{(s)}$ all the same, or allowing them to be separate which allows us to train more parameters per speaker, and it offers a convenient way of applying multiple constrained MLLR transforms. We envisage using a relatively small value of K, for example less than 20. Note that as K approaches I the basic model loses its power, since we no longer have any correlations to model if there is only one Gaussian in each "sub-model". However as we will describe later we can still make use of the correlations across sub-models through the use of an appropriate prior over the vectors $\mathbf{v}_{jkm}$ in a state.

5. RELATIONSHIP TO PREVIOUS WORK

In terms of speech recognition work, this technique probably has the most similarity to Eigenvoices [4] and Cluster Adaptive Training and Extended SAT [5]. However, there is a very significant difference because both of those techniques are focused on speaker adaptation; they both attempt to compactly represent the most significant dimensions of variation between speakers. In this work, we use similar techniques to represent the variation between phonetic speech states, and only secondarily apply the same ideas to modeling speaker variation. The best reference point for this work is probably to be found in the speaker identification literature. The two-factor approach of Equation (11) is very similar to factor analysis as used in [3] for speaker identification. In that work, the two factors are the speaker and the channel, with the channel being a "nuisance factor". In our case the factor of interest is the phonetic state and the "nuisance factor" is the speaker plus channel. Our model differs in a number of respects from the one used in [3]. We model the weights of the shared Gaussians, which is not generally done in speaker identification. But we do not attempt to marginalize over any of the parameters as is done in [3], which we believe would make very little difference and would make the model hopelessly inefficient. We also do not include the "diagonal term" used there, which essentially amounts to putting a Maximum A Posteriori (MAP) estimation of the full set of mean parameters of the GMM on top of the subspace. This would introduce a potentially vast number of parameters which would have to be heavily pruned in order to fit in memory, and would significantly complicate training. Other extensions which we have added to the basic scheme include "sub-states" and "sub-models", and we have also done some work to devise a subspace form of constrained MLLR estimation and integrate it with this type of model. The estimation process and the methods used for fast computation also differ.

Our own previous work along these lines started with [6] in which we used a shared structure of GMM together with Maximum A Posterior (MAP) estimation to model speech states. The tree structure of the clustered speech states was used in the MAP estimation. That approach gave substantial improvements with only Maximum Likelihood training, but used a very large number of parameters which would have made discriminative training infeasible. In [1] (a book chapter, written but not yet published at the time of writing) we give a few more results on that work and also the approach we are currently describing. We favor the current approach since it has a much smaller number of parameters so it is feasible to extend it to discriminatively trained systems (we report discriminatively trained results in [1]), and it gives better results than the MAP-based scheme even under ML estimation when the amount of training data is limited. The technical details which were omitted in [1] because of space constraints are given in the technical report [2]. The details of training are given there in a less complete form than the current document, and describe a slightly different recipe than what is given here, corresponding to the experiments we reported in [1]. That recipe included discriminative training, but did not include sub-models, the associated use of priors, multi-class constrained MLLR or the subspace version of constrained MLLR. It also used a more complicated model initialization scheme than what we describe here, and used MLLR which is very inefficient when combined with these types of models (and gives very little additional improvement).

The results reported in [1] showed that this type of model is able to beat a conventionally structured HMM in an evaluation setting (i.e. comparing against the very best system we can build), but only by a very small margin given the recipe we were using at the time. Based on results we reported there on smaller amounts of training data (50 hours, as opposed to thousands of hours), we believe that the advantages of this approach will be much clearer when the quantity of training data is relatively small. This approach also gives more improvement when no discriminative training is done, which again is relevant to an environment where resources are limited, and it has a

large proportion of its parameters not tied to any speech state, which enables the use of out-of-domain data in a natural way, i.e. to help train those generic parameters.

6. SUMMARY OF TRAINING OF SUBSPACE GMM

The training of this style of model is more complicated than normal GMM training as the model has more different types of parameters. In this section we attempt to give an overview of the process prior to the more detailed treatment that will be given in later sections. The basic idea is that after initializing all the parameters of the model in a sensible way, we repeatedly pick a parameter type and optimize that parameter given the others. The optimization of parameters takes place through standard Estimation Maximization approaches. Some types of parameters can be optimized simultaneously, and some cannot. We do not specify the exact order in which parameters should be updated; this is a matter for experimentation.

6.1. Overview of re-estimation

The typical procedure for model initialization and training will be:

Initialize background GMM.
Initialize model.
For each iteration:
  Choose the subset of types of parameters we will update.
  For each speaker s:
    Optionally, reset the speaker factors and transforms to zero
    For zero or more speaker-adaptation iterations:
      Accumulate either speaker-factor or speaker-transform statistics.
      Do the appropriate update
    Accumulate the appropriate subset of types of global statistics
  Update the selected types of parameters
  Optionally do "mixing up"– increase the number of sub-states.
  Optionally increase the subspace dimension S or T.

6.2. Types of re-estimation

Here we review the types of parameters we will be re-estimating in this model and discuss which of them we can combine on the same iteration.

The types of model parameters to be re-estimated (excluding parameters relating to constrained MLLR) are:

- Model vectors and weights $\mathbf{v}_{jkm} \in \Re^S$ and $c_{jkm} \in \Re$
- Model projections $\mathbf{M}_{ki} \in \Re^{D \times (S+1)}$
- Speaker projections $\mathbf{N}_{ki} \in \Re^{D \times (T+1)}$
- Weight projections $\mathbf{w}_{ki} \in \Re^{S+1}$
- Within-class variances $\mathbf{\Sigma}_{ki} \in \Re^{D \times D}$
- Background model means $\bar{\boldsymbol{\mu}}_{ki} \in \Re^D$ and variances $\bar{\mathbf{\Sigma}}_{ki} \in \Re^{D \times D}$ and weights $\bar{w}_{ki} \in \Re$.

There are also per-speaker parameters; these relate to speaker adaptation and are not normally considered part of the model:

- Speaker factors $\mathbf{v}_k^{(s)} \in \Re^T$
- Speaker transforms $\mathbf{W}_k^{(s)} \in \Re^{D \times (D+1)}$

We will probably do these two kinds of re-estimation on separate passes over each speaker rather than trying to combine them. We have a choice as to whether to reset the speaker factors and/or transforms to zero each time we do the update of the global parameters, so we can start estimating them from scratch each time. This would probably not reach as good a training likelihood but it might be a better match to test time when we will presumably do a small number of iterations of adaptation for each speaker.

There are also globally shared (non-speaker-specific) parameters that relate to the use of constrained MLLR with parameter subspaces. They relate to the framework for constrained MLLR estimation which will be described in Sections 12 and 13. They are:

- Initialization of pre-transform and mean scatter for speaker transforms: $\mathbf{W}_{\text{pre}} \in \Re^{D \times (D+1)}$, $\mathbf{D} \in \Re^{D \times D}$ (diagonal), and sub-model specific versions $\mathbf{W}_{\text{pre}}^{(k)}$, $\mathbf{D}_k$.
- Computation of speaker transform basis elements: global basis matrices $\tilde{\mathbf{W}}_b \in \Re^{D \times (D+1)}$, $1 \leq b \leq B$, and sub-model specific basis matrices $\tilde{\mathbf{W}}_b^{(k)}$.

These parameters will be properly introduced starting in Section 12. Unlike the other globally shared parameters they are not subject to any form of iterative re-estimation but are estimated just once, typically after at least a few iterations of model estimation.

6.3. Constraints on combining updates

There are certain constraints on which of the updates above we can combine on a single iteration. The only practical problem involves the first three items in the list above. Theoretically, if we update any of the three we cannot show that the update for any other of the three will increase the data likelihood, given the update formulae we use. We believe that practically speaking we can combine any two of them, or any three if we introduce an arbitrary constant $\nu < \frac{2}{3}$ that interpolates between the original parameter values and the updated ones, i.e. multiplies the step size by ν. Experience seems to bear this out. The reasoning is as follows. All of these types of updates are essentially finding the solution of a quadratic objective function. When we combine several such updates, the danger is that some of the different classes of parameters are doing the "same thing" or are effectively the same parameter, and are being updated too many times, leading to a learning rate that is too high. But any update that is less than twice as fast as the "ideal" update that just jumps to the solution of the quadratic objective function, will still converge. This can easily be checked for a scalar quadratic objective function. It is when we go above two "non-combinable" updates that we anticipate practical problems. That is why we need to introduce a constant ν that interpolates the old and new parameters to bring down the effective "normalized" learning rate for any effectively shared parameters to less than two.

6.4. Warning on update order

In instances where we want to combine global updates, for instance we want to combine updating the vectors $\mathbf{v}_{jkm}$ with the model projections $\mathbf{M}_{ki}$, we will have situations where the update formula for one parameter type refers to the other parameter type. In that case it is important in some cases to use the pre-update version of the other type of parameter. The way we will make this clear is to use a hat (e.g. $\hat{x}$) on newly updated parameters, so we might write something that looks like:

$$\hat{y}_i = y_i + s_i \tag{15}$$
$$\hat{x}_i = x_i + t_i y_i. \tag{16}$$

In this case the parameters with a hat will refer to the next iteration's values and we will also understand that any types of parameter we are not updating will just be copied from one iteration to the next. However we slightly abuse this hat notation at times; we will try to explain in the text what is meant. We avoid introducing iteration indices as we already have a proliferation of indices, and some the updates have nested loops which would require an index for each loop. The reason why it is sometimes important to keep track of whether the update refers to the old or updated version of the other parameter, is that in many cases the statistics for one parameter will contain expressions that include the other parameter. Then, if in the update formula for one parameter we use the updated version of the other parameter it will be inconsistent with the one that is implicitly the statistics and the update becomes invalid.

7. INITIALIZING AND PRE-TRAINING THE BACKGROUND GMM

In the techniques we are using, we need to start out with a generic Gaussian Mixture Model (GMM) that models all speech and silence. We call this the "background GMM", although this term is used in a slightly different sense than in the speaker identification literature. This GMM will typically have around 500 to 1000 Gaussians in it. In the recipe envisaged here, these would be full covariance Gaussians but we also keep around the diagonalized versions of them for purposes of Gaussian selection for fast computation. These Gaussians will be trained on some kind of baseline features, e.g. MFCC+Δ+$\Delta\Delta$, possibly with adaptation applied (VTLN, constrained MLLR). The initialization and phoneme-independent re-estimation of these Gaussians is something that we cannot apply a lot of theory to, and in this section we review the kinds of techniques that we intend to use for initializing and "pre-training" the mixture of Gaussians. The way this has been done in previous experiments is to initialize a mixture of diagonal Gaussians by clustering the Gaussians from a baseline system, and to re-estimate them as full covariance Gaussians on a subset of the data. We can also consider training them from scratch, which may involve less code and does not have the dependency on the baseline system. Something else we can try (which has not been done before) is to alternate iterations of training this full-covariance GMM with iterations of re-estimating constrained MLLR transforms per speaker. This means that the trained GMM will be a "speaker adaptively trained" GMM and we have a natural way of computing the constrained MLLR transforms without doing a first pass of recognition. The recipe we give in the rest of this document does not include this, although it does not require any new techniques.

7.1. Initializing the GMM from a trained, diagonal system

When initializing the GMM we start with a set of diagonal Gaussians derived from a baseline HMM set. Let the Gaussians $1 \leq j \leq J$ have means $\boldsymbol{\mu}_j$, weights w_j and diagonal covariances $\boldsymbol{\Sigma}_j$. J will typically be in the tens of thousands. Note that this J is not the same as the J we have used previously to represent the number of clustered states. Let the dimension be $1 \leq d \leq D$ (e.g. $D = 40$) so the mean's d'th dimension will be $\boldsymbol{\mu}_{jd}$ and the variance's j'th dimension will be σ_{jd}^2. (This is by convention that lower-case σ refers to a standard deviation so we need to square it to get a variance; we could equivalently use the notation $(\boldsymbol{\Sigma}_j)_{ii}$). The weights w_j could just be the weights of Gaussians within the individual HMM states of the original system, renormalized to sum to one. If the counts of the states were available these could be used, but it might somewhat bias the resulting mixture towards silence which might not be good.

7.1.1. Clustering

We will use the notation $\bar{\boldsymbol{\mu}}_i$ and $\bar{\boldsymbol{\Sigma}}_i$ to represent the means and variances of the background GMM, for $1 \leq i \leq I$ with for example $I = 750$ typically. The "background" GMM is a generic GMM that has not yet been adapted to each state. We also define weights $\bar{w}_i$ for each of the background Gaussians, but it may be best to set these to be all the same, in order to encourage the Gaussians to all get similar occupation counts. In Appendix E we describe an algorithm to take a large number of diagonal Gaussians and cluster them to a smaller number. We will use this to initialize the background GMM, starting from a very large GMM that we obtain by taking all of the Gaussians in a baseline system, putting all of them into a single mixture and renormalizing the weights to sum to one. At this point we cluster the Gaussians down to I clusters, each represented by a diagonal Gaussian with a mixture weight, using the algorithm described in Appendix E. We may choose at this point to make the mixture weights all equal to $1/I$ in order to encourage even distribution of counts among the cluster centers during further training. Later we will train these Gaussians as full-covariance Gaussians on unlabeled training data.

7.1.2. Super-clustering

The use of sub-models as described in Section 4.6 requires that we cluster the first level of clustered Gaussians into another level of clusters, so we take the initial I Gaussians and cluster them using the same algorithm into K clusters each of size I_k. It might make sense to do the super-clustering after re-estimation of the Gaussians derived from the clusters as described in Section 7.2.

7.1.3. Minimum size of clusters

If we want to enforce a minimum size on clusters (e.g. when we do super-clustering), it is probably most practical to enforce this in a soft manner as follows. If the "soft" minimum cluster size is M, we can use a constant k (e.g. $k = 0.1/J$) and add a "penalty term" to the objective function of: $\sum_{i=1}^{I} k \min(0, |\mathcal{S}_i| - M)^2$. It is easy to incorporate this into the algorithm above, as whenever we consider moving a point from cluster i to i' we can calculate the value of this extra term for i and i' before and after the move, and add it to the likelihood difference. This approach will stop the minimum cluster size from being much smaller than M; it would be more complicated to enforce a hard limit without either affecting the cluster quality or the speed of the algorithm.

7.2. Re-estimating the background GMM prior to training

The proposed recipe will allow the re-estimation of the background GMM through E-M on some generic speech data, prior to training the model itself. We would have to do experiments to see whether this actually helps. Note that we may also re-estimate the background GMM during the main training procedure itself, but that is a separate issue from what we are discussing here.

The statistics accumulation for re-estimating the background GMM is quite easy and we will not write down the equations. Firstly, just a note on the fast computation of posteriors in the background GMM. More details will be given in Section 9.2. As noted above, we store the diagonal inverse variances, and the first thing we do on each frame is to compute the contribution of all the diagonal

Gaussians' probabilities to the frame's likelihood and sort these to select e.g. the 50 most likely indexes i. The full-variance likelihood computation is then done using only these preselected indices. The full-variance statistics will be accumulated given the resulting posteriors. A further stage of pruning takes place before we accumulate any statistics so that we can avoid using very tiny counts.

The re-estimation formulae for the full variance Gaussians are too obvious to write here. Various extra details should be noted, though. On each iteration of update we also store the diagonal of the re-estimated variance, for purposes of fast likelihood evaluation. We may set the weights to be all the same rather than using the normal formula. If we encounter an index i for which the data count is too small (e.g. less than twice the dimension d), an easy thing to do is to set the updated mean and variance for index i to be equal to the *pre*-update version of the mean and variance for some other index, e.g. $i+1$. Choosing the pre-update value ensures that it will differ from the post-update version of the selected other index. For some data sources, a significant amount of the data will be linearly dependent and this can lead to singular covariance matrices. A suitable means of preventing this is to floor the covariance matrices to some small multiple of the global covariance. A process for flooring symmetric matrices is described in Appendix I.

7.3. Overall order of preparing the background GMM

Here we summarize the procedures we intend to use to prepare the background GMM prior to training the main model. We describe the most general procedure with all the bells and whistles; some of these are optional (in particular, the speaker adaptation).

1. Cluster the Gaussians a baseline model to I clusters; each cluster is a diagonal Gaussian.
2. For several iterations (e.g. 3 or 4):
   - For each speaker:
     - Optionally re-estimate constrained MLLR transform for the speaker
     - Accumulate statistics for full-covariance GMM parameter update
   - Update full-covariance GMM parameters.
   - Possibly set all weights to be the same after re-estimation.
   - Make diagonal GMM parameters as (diagonalized) copy of full ones.
3. Cluster the I Gaussians to K super-clusters.

8. INITIALIZATION OF MODEL

In this section we describe the initialization of the main model.

8.1. Overview of model initialization

The procedure we describe here for model initialization is different from that described in [2]. In that document, we accumulated pruned count and mean statistics over the product of HMM states j by Gaussian indices i and used them in an iterative update procedure in memory to update the model vectors $\mathbf{v}_{jm}$ (there were no sub-models k at that time) and transforms $\mathbf{M}_i$ and $\mathbf{w}_i$, starting from random initialization of the model vectors. The disadvantage of that approach was that it involved a substantial amount of extra coding as it required a whole parallel accumulation and update apparatus and a parallel statistics format. Here we describe a much simpler approach that should reach the same likelihood values with only one or two extra iterations over the data. Something that we should probably investigate is whether it is important to first train to convergence with the Gaussian posteriors of the original background GMM (which is effectively what we were doing with the old approach). This can easily be simulated in the current setup by using very tight pruning beams when we prune using the background model, or by directly using the Gaussian posteriors of the background model for the first few iterations of training which can be introduced as an option in the code (i.e. set $\gamma_{jkmi}(t) = \frac{\mathcal{N}(\mathbf{x}_t;\bar{\boldsymbol{\mu}}_{ki},\bar{\boldsymbol{\Sigma}}_{ki})}{\sum_{k,i}\mathcal{N}(\mathbf{x}_t;\bar{\boldsymbol{\mu}}_{ki},\bar{\boldsymbol{\Sigma}}_{ki})}$).

The model initialization procedure that we describe here does not allow us to make the subspace size greater than the feature dimension. To get larger sizes we can increase it at a later stage.

8.2. Feature normalizing transform

Prior to the main model initialization we need to obtain a feature normalizing transform that will make the within class variance unit and the between class variance diagonal. This will be used in the initialization of or in increasing the dimension of the projections $\mathbf{M}_{ki}$ and $\mathbf{N}_{ki}$. It can be derived from the parameters of the background GMM during the initialization of the main model. We compute the within-class and between-class variance as follows (assuming $\sum_{i=1}^{I}\bar{w}_i = 1$):

$$\boldsymbol{\Sigma}_W = \sum_{i=1}^{I}\bar{w}_i\bar{\boldsymbol{\Sigma}}_i \tag{17}$$

$$\boldsymbol{\mu} = \sum_{i=1}^{I}\bar{w}_i\bar{\boldsymbol{\mu}}_i \tag{18}$$

$$\boldsymbol{\Sigma}_B = \left(\sum_{i=1}^{I}\bar{w}_i\bar{\boldsymbol{\mu}}_i\bar{\boldsymbol{\mu}}_i^T\right) - \boldsymbol{\mu}\boldsymbol{\mu}^T \tag{19}$$

We want a transformation that makes $\boldsymbol{\Sigma}_W$ unit and diagonalizes $\boldsymbol{\Sigma}_B$. We first do the Cholesky decomposition $\boldsymbol{\Sigma}_W = \mathbf{L}\mathbf{L}^T$, compute $\mathbf{S} = \mathbf{L}^{-1}\boldsymbol{\Sigma}_B\mathbf{L}^{-T}$, and do the singular value decomposition $\mathbf{S} = \mathbf{U}\mathbf{D}\mathbf{V}^T$. Since $\mathbf{S}$ is symmetric positive semi-definite this implies $\mathbf{S} = \mathbf{U}\mathbf{D}\mathbf{U}^T$ (see Appendix B). It should be verified that the diagonal elements of $\mathbf{D}$ and the corresponding columns of $\mathbf{U}$ are sorted by decreasing eigenvalue. The transformation we want is then $\mathbf{T} = \mathbf{U}^T\mathbf{L}^{-1}$. This transform should be recorded as it will be needed later.

8.3. Initialization if offsets are used

The initialization in the case that we plan to use offsets on the vectors (i.e. if we are using terms like $\mathbf{v}^+$) is as follows. We require that $S \leq D$ and $T \leq D$.

$$M_{jk} = 1 \tag{20}$$

$$c_{jk1} = \frac{1}{K} \tag{21}$$

$$\mathbf{v}_{jk1} = \mathbf{0} \in \Re^S. \tag{22}$$

$$\mathbf{M}_{ki} = \left[\left(\mathbf{T}^{-1}\right)_{1:D,1:S} \; ; \; \bar{\boldsymbol{\mu}}_{ki}\right] \tag{23}$$

$$\mathbf{N}_{ki} = \left[\left(\mathbf{T}^{-1}\right)_{1:D,1:T} \; ; \; \mathbf{0}\right] \tag{24}$$

$$\mathbf{w}_{ki} = \mathbf{0} \in \Re^{S+1}. \tag{25}$$

The notation $\left(\mathbf{T}^{-1}\right)_{1:D,1:S}$ means the first S columns of $\mathbf{T}^{-1}$.

8.4. Initialization if no offsets are used

If no offsets will be used on the vectors we need to initialize the vectors to a nonzero value, and we choose to put this in the first dimension. Note that below, $\mathbf{e}_1$ is a unit vector in the first dimension, i.e. $[\,1\;0\;\ldots\;0]$. We require that $S \leq D+1$ and $T \leq D$.

$$M_{jk} = 1 \quad (26)$$

$$c_{jk1} = \frac{1}{K} \quad (27)$$

$$\mathbf{v}_{jk1} = \mathbf{e}_1 \in \Re^S. \quad (28)$$

$$\mathbf{M}_{ki} = \left[\, \bar{\boldsymbol{\mu}}_{ki} \,;\, \left(\mathbf{T}^{-1}\right)_{1:D,1:S-1} \right] \quad (29)$$

$$\mathbf{N}_{ki} = \left(\mathbf{T}^{-1}\right)_{1:D,1:T} \quad (30)$$

$$\mathbf{w}_{ki} = \mathbf{0} \in \Re^S. \quad (31)$$

8.5. Increasing the subspace dimension

It is possible to increase the phonetic or speaker subspace dimension after some iterations of training. Typically a subspace dimension slightly larger than the feature dimension is optimal. We describe this here as it is a similar process to the initialization, although it would take place during model update. Note that we must wait at least 2 iterations before doing this, as we need to wait for the matrices $\mathbf{M}_{ki}$ and $\mathbf{N}_{ki}$ to deviate from their initial values: they do not do so on the first iteration because the vectors have not yet been trained at that point.

Now we describe how to increase the subspace dimension if we are using offset features $\mathbf{v}^+$. Assuming we are increasing the model subspace dimension from S to S' (and $S'-S$ cannot exceed D), we will have (if using offsets):

$$\hat{\mathbf{M}}_{ki} = \left[\mathbf{m}_{ki}(1)\ldots\mathbf{m}_{ki}(S) \;\; \left(\mathbf{T}^{-1}\right)_{1:D,1:(S'-S)} \;\; \mathbf{m}_{ki}(S+1)\right], \quad (32)$$

where $\mathbf{m}_{ki}(s)$ is the s'th column of $\mathbf{M}_{ki}$, and with $\mathbf{T}$ as computed in Section 8.2. We have a similar thing for the speaker transform:

$$\hat{\mathbf{N}}_{ki} = \left[\mathbf{n}_{ki}(1)\ldots\mathbf{n}_{ki}(T) \;\; \left(\mathbf{T}^{-1}\right)_{1:D,1:(T'-T)} \;\; \mathbf{n}_{ki}(T+1)\right]. \quad (33)$$

At this point we also need to extend the model and speaker vectors $\mathbf{v}_{jkm}$ and $\mathbf{v}_k^{(s)}$ by appending the appropriate number of zeros $S'-S$ or $T'-T$, and extend the weight projections $\mathbf{w}_{ki}$ by inserting $S'-S$ zeros just before the final element.

If not using offsets, the step of increasing the dimensions is a little simpler:

$$\hat{\mathbf{M}}_{ki} = \left[\mathbf{M}_{ki} \;\; \left(\mathbf{T}^{-1}\right)_{1:D,1:(S'-S)}\right] \quad (34)$$

$$\hat{\mathbf{N}}_{ki} = \left[\mathbf{N}_{ki} \;\; \left(\mathbf{T}^{-1}\right)_{1:D,1:(T'-T)}\right]. \quad (35)$$

In this case we extend the model and speaker vectors $\mathbf{v}_{jkm}$ and $\mathbf{v}_k^{(s)}$ by appending the appropriate number of zeros $S'-S$ or $T'-T$, and the weight projections $\mathbf{w}_{ki}$ would also be extended by appending $S'-S$ zeros at the end and not by inserting them before the final element.

8.6. Initializing the within-class variances

The within-class variances $\boldsymbol{\Sigma}_{ki}$ are initialized to be equal to the full-covariance background GMM's within-class variances

$$\boldsymbol{\Sigma}_{ki} = \bar{\boldsymbol{\Sigma}}_{ki}. \quad (36)$$

8.7. Initializing the speaker transforms

The speaker transforms $\mathbf{W}_k^{(s)}$ are all initialized to the "default" transform $\mathbf{I}^-_{D+1}$, by which we mean the identity matrix of size $D+1$ with the last row removed. Typically the speaker transforms would not be stored centrally but would be generated in memory for each training or test speaker, so we would not have to do this.

9. LIKELIHOOD EVALUATION

In this section we describe the model likelihood computation procedure, which is needed both in decoding and statistics accumulation.

9.1. Global and speaker-specific pre-computation

Prior to seeing any feature data there are some quantities that need to be pre-computed. There are the per-Gaussian normalizers:

$$n_{jkmi} = \log c_{jkm} + \log w_{jkmi} - 0.5(\log\det\boldsymbol{\Sigma}_{ki} + D\log(2\pi) + \boldsymbol{\mu}_{jkmi}^T\boldsymbol{\Sigma}_{ki}^{-1}\boldsymbol{\mu}_{jkmi}) \quad (37)$$

with:

$$\boldsymbol{\mu}_{jkmi} = \mathbf{M}_{ki}\mathbf{v}_{jkm}^+. \quad (38)$$

These normalizers take long enough to compute that it is worthwhile storing them on disk, although they should be in a separate file from the model parameters because they take up a lot of space and can easily be regenerated. In our previous work in [1, 2], these were computed in parallel and combined into a single file on disk. However, we do not believe it is necessary to compute them in parallel here since there are are certain steps in the update which without additional optimizations and approximations (which we have not described here) will take as long as computing the normalizers. This should be a manageable amount of time (a few minutes) as long as the models are reasonably small or are broken up via the use of sub-models. We also need to compute the speaker-specific offsets:

$$\mathbf{o}_{ki}^{(s)} = \mathbf{N}_{ki}\mathbf{v}_k^{(s)+}, \quad (39)$$

and the log determinants of the per-speaker constrained MLLR transforms:

$$\mathrm{logdet}_k^{(s)} = \log|\det\mathbf{A}_k^{(s)}|, \quad (40)$$

where $\mathbf{A}_k^{(s)}$ is the square part of transform $\mathbf{W}_k^{(s)}$.

9.2. Gaussian selection

The first stage in the process on each frame is Gaussian selection, in which we use first the diagonal version of the background model and then the full-covariance version, to select a set of Gaussian indices to limit our further computation. The set of selected indices will be a set of pairs (k,i) reflecting the two level structure of the model with sub-models. We have two forms of adaptation available: the constrained MLLR transforms $\mathbf{W}_k^{(s)}$ and the speaker vectors $\mathbf{v}_k^{(s)}$. We have a choice as to whether to apply one or both of these to the two phases of Gaussian selection (diagonal and full). The mathematics below assumes we apply both forms of adaptation to both phases, but this can be experimented with.

Prior to Gaussian selection we pre-compute speaker adapted features for each sub-model index k:

$$\mathbf{x}_k(t) = \mathbf{W}_k^{(s)}\mathbf{x}^+(t). \quad (41)$$

During Gaussian selection we will also compute on the fly the following quantity, which reflects the factor-based adaptation:

$$\mathbf{x}_{ki}(t) \quad = \quad \mathbf{x}_k(t) - \mathbf{o}_{ki}^{(s)}, \tag{42}$$

with $\mathbf{o}_{ki}^{(s)}$ as computed in Equation (39).

The process of Gaussian selection is as below, with for example pruning parameters $P^{\text{diag}} = 50$ and $P = 10$. Note that we have a diagonal version of the background model with mean and variance $\bar{\boldsymbol{\mu}}_{ki}^{\text{diag}}$ and and $\bar{\boldsymbol{\Sigma}}_{ki}^{\text{diag}}$. This is primarily used as a speedup for the full-covariance model, and the mean potentially differs as well as the variance because we may train the two models with different amounts of adaptation.

For $k = 1 \dots K$,
  For $i = 1 \dots I_k$, compute:
    $\log p^{\text{diag}}(\mathbf{x}(t), k, i) = \text{logdet}_k^{(s)} + \log \bar{w}_{ki} \dots$
      $+ \log \mathcal{N}(\mathbf{x}_{ki}(t) | \bar{\boldsymbol{\mu}}_{ki}^{\text{diag}}, \bar{\boldsymbol{\Sigma}}_{ki}^{\text{diag}})$.
Prune to the P^{diag} pairs (k, i) with highest $\log p^{\text{diag}}(\mathbf{x}(t), k, i)$
For each of these top pairs, compute:
  $\log p(\mathbf{x}(t), k, i) = \text{logdet}_k^{(s)} + \log \bar{w}_{ki} \dots$
    $+ \log \mathcal{N}(\mathbf{x}_{ki}(t) | \bar{\boldsymbol{\mu}}_{ki}, \bar{\boldsymbol{\Sigma}}_{ki})$
Prune to the P pairs (k, i) with highest $\log p(\mathbf{x}(t), k, i)$

9.3. Pre-computation per frame

After Gaussian selection there are certain quantities that should be computed for each of the selected pairs of indices (k, i). We compute and store:

$$\mathbf{x}_{ki}(t) \quad = \quad \mathbf{x}_k(t) - \mathbf{o}_{ki}^{(s)} \tag{43}$$

$$\mathbf{z}_{ki}(t) \quad = \quad \mathbf{M}_{ki}^T \boldsymbol{\Sigma}_{ki}^{-1} \mathbf{x}_{ki}(t) \tag{44}$$

$$n_{ki}(t) \quad = \quad \text{logdet}_k^{(s)} - 0.5 \mathbf{x}_{ki}(t)^T \boldsymbol{\Sigma}_{ki}^{-1} \mathbf{x}_{ki}(t) + z_{ki}(t)_{(S+1)}. \tag{45}$$

so $\mathbf{x}_{ki}(t)$ is the "speaker-adapted" version of the features used for Gaussian index (k, i), $\mathbf{z}_{ki}(t)^-$ is like a "covector" to quantities $\mathbf{v}_{jkm}$ (we will dot them to get the linear part of the likelihood), and $n_{ki}(t)$ is a normalizer per frame that contains terms independent of the model.

9.4. Gaussian likelihood computation

We can compute the contribution to the likelihood from state j, mixture m and Gaussian index k, i as:

$$\log p(\mathbf{x}(t), k, m, i | j) = n_{ki}(t) + n_{jkmi} + \mathbf{z}_{ki}(t)^- \cdot \mathbf{v}_{jkm}. \tag{46}$$

When doing the computation for a particular state j, we will iterate over the preselected set of P pairs (k, i), and then for $1 \leq m \leq M_{jk}$ we will compute the above quantity. We will accumulate an array of the tuples $(k, i, m, \log p(\mathbf{x}(t), k, m, i | j))$ and do pruning such that if some element has probability much less than the best (e.g. a beam of 5), we discard it. It may be helpful to avoid adding elements to the array in the first place if they are lower than the current maximum minus the specified beam.

We then compute the total likelihood as:

$$\log p(\mathbf{x}(t) | j) = \log \sum_{k,m,i} p(\mathbf{x}(t), k, m, i | j). \tag{47}$$

Note that typically these kinds of summations are 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, in case the floating point range is too small.

10. ACCUMULATION

10.1. Pruned posterior computation

All of the forms of accumulation require us to compute posteriors over the individual Gaussians, each represented by the 4-tuple of indices (j, k, i, m). This can be done using the state likelihoods $\log p(\mathbf{x}(t) | j)$ and the per-Gaussian likelihoods $p(\mathbf{x}(t), k, m, i | j)$ described in the previous section. The posteriors of Gaussians are given as follows:

$$\gamma_{jkmi}(t) \quad \equiv \quad p(j, k, m, i | \mathbf{x}(t)) \tag{48}$$

$$\quad = \quad p(j | \mathbf{x}(t)) \frac{p(\mathbf{x}(t), k, m, i | j)}{p(\mathbf{x}(t) | j)}, \tag{49}$$

where $p(j, k, m, i | \mathbf{x}(t))$ and $p(j | \mathbf{x}(t))$ are as defined in Equations (46) and (47), and $p(j | \mathbf{x}(t)) \equiv \gamma_j(t)$ will typically be supplied to the module that does these computations, e.g. it will be a zero or one posterior derived from Viterbi alignment, or will be derived from some kind of forward backward algorithm. In order to get a good initialization, state posteriors $\gamma_j(t)$ derived from a baseline system should be used for at least the first few iterations of training.

We use (49) to compute posteriors of the pruned list of tuples (j, k, m, i). We then do a further stage of pruning. Most of the accumulation steps will require more computation than the dot product that was required to compute the likelihoods in (46), so it is worthwhile to prune more but we want to preserve expectations during estimation. First we decide on a minimum posterior value, say $f = 0.125$. Then we compute pruned posterior values:

$$\tilde{\gamma}_{jkmi}(t) \; = \; \text{randprune}(\gamma_{jkmi}(t), f) \tag{50}$$

$$\text{randprune}(x, f) \; = \; \begin{cases} x \geq f \rightarrow x \\ x < f \rightarrow \begin{cases} f \text{ with probability } \frac{x}{f} \\ 0 \text{ otherwise} \end{cases} \end{cases} \tag{51}$$

We will use these pruned posterior values in statistics accumulation.

10.2. A note on storing posteriors compactly

It is possible to store the posteriors more compactly than using floating point values. We describe this here but it is not recommended for a basic implementation. We can use a modified version of the randomized pruning described above to express all posteriors as an integer multiple of f. This can be combined with compression techniques that compress small values in memory– a convenient method is to store a count in one character, using positive values to store the actual counts and negative values to store offsets into a separate resizable array for "overflow" values larger than 127. If we keep a separate resizable array for every 128 elements in the overall array of integer values we are compressing we will always be able to store the index, so using this method it is possible to store integers in very little more than 1 byte, assuming most of the integers have values less than 128.

We do not anticipate that this will be essential– the use of "submodels" (index k) reduces the amount of count statistics, and anyway (for ML training) the count statistics only take the same amount of memory as the normalizers which we have to store anyway, so the

normalizers would also have to be compressed (in fact, for the work reported in [1, 2], we also compressed the normalizers in a quite similar way).

10.3. Counts accumulation

The count statistics should be computed on all iterations as they appear in the updates of most of the parameter types:

$$\gamma_{jkmi} = \sum_t \tilde{\gamma}_{jkmi}(t). \tag{52}$$

10.4. Model vectors accumulation

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

$$\mathbf{y}_{jkm} = \sum_{t,i} \tilde{\gamma}_{jkmi}(t)\mathbf{z}_{ki}(t)^{-}, \tag{53}$$

with $\mathbf{z}_{ki}(t)$ as given by Equation (44), and $\cdot^{-}$ refers to removing the last vector element. This is the linear term in the auxiliary function for $\mathbf{v}_{jkm}$; the quadratic term can be worked out from the counts.

10.5. Model projections accumulation

The sufficient statistics for the model projections $\mathbf{M}_{ki}$ are

$$\mathbf{Y}_{ki} = \sum_{t,j,m} \tilde{\gamma}_{jkmi}(t)\mathbf{x}_{ki}(t){\mathbf{v}_{jkm}^{+}}^{T}. \tag{54}$$

If we left-multiply this by $\mathbf{\Sigma}_{ki}^{-1}$ it is the linear term of the auxiliary function in $\mathbf{M}_{ki}$, but it is more convenient to do that multiplication during the update. These statistics are also needed for the re-estimation of the within-class variances $\mathbf{\Sigma}_{ki}$.

10.6. Speaker vectors accumulation

The accumulation needed to re-compute the per-speaker vectors $\mathbf{v}_k^{(s)}$ is analogous to the model vectors accumulation in Section 10.4. We first define a speaker-space analogue to $\mathbf{x}_{ki}(t)$, in which we treat the main model as an offset:

$$\mathbf{x}_{jkmi}(t) = \mathbf{x}_k(t) - \boldsymbol{\mu}_{jkmi}, \tag{55}$$

with the un-speaker-adapted mean $\boldsymbol{\mu}_{jkmi}$ as defined in Equation (38). It may be helpful during statistics accumulation to cache this quantity $\boldsymbol{\mu}_{jkmi}$ on each frame for which the pruned posterior $\tilde{\gamma}_{jkmi}$ is nonzero, since it appears in several expressions. It cannot be precomputed globally because it would take up too much memory; caching is possible but would not affect the speed of the algorithm very much. We then need to define a speaker-subspace analogue to the "co-vector" $\mathbf{z}_{ki}(t)$ which we defined in Equation (44). We have it with:

$$\mathbf{z}_{jkmi}(t) = \mathbf{N}_{ki}^T\mathbf{\Sigma}_{ki}^{-1}\mathbf{x}_{jkmi}(t). \tag{56}$$

It will be useful to pre-compute the quantity $\mathbf{N}_{ki}^T\mathbf{\Sigma}_{ki}^{-1}$ before this type of accumulation. The statistics are accumulated below, where $\mathcal{T}(s)$ is the set of frames that cover the data for speaker s:

$$\gamma_{ki}^{(s)} = \sum_{t\in\mathcal{T}(s),j,m} \tilde{\gamma}_{jkmi}(t) \tag{57}$$

$$\mathbf{y}_k^{(s)} = \sum_{t\in\mathcal{T}(s),i,j,m} \tilde{\gamma}_{jkmi}(t)\mathbf{z}_{jkmi}(t)^{-}. \tag{58}$$

10.7. Speaker projections accumulation

Here are the statistics to update the speaker projections $\mathbf{N}_{ki}$. (Note that this is a global type of parameter, not a speaker specific one). These are analogous to the statistics $\mathbf{Y}_{ki}$ for the model projections $\mathbf{M}_{ki}$:

$$\mathbf{Z}_{ki} = \sum_{t,j,m} \tilde{\gamma}_{jkmi}(t)\mathbf{x}_{jkmi}(t){\mathbf{v}_k^{(s(t))}}^{+T}, \tag{59}$$

where we write $s(t)$ to mean the speaker active on frame t. We also have to accumulate a weighted outer product of the speaker vectors:

$$\mathbf{R}_{ki} = \sum_{s,k}\left(\sum_{t\in\mathcal{T}(s),j,m}\tilde{\gamma}_{jkmi}(t)\right)\mathbf{v}_k^{(s)^{+}}{\mathbf{v}_k^{(s)}}^{+T}, \tag{60}$$

and it would be most efficient to only update the matrix once per speaker using cached counts, although this is not very critical. This quantity is symmetric so only the lower triangular part should be stored.

10.8. Statistics for within-class variances and full covariance background model

The following statistics are needed in order to update the within-class variances $\mathbf{\Sigma}_{ki}$ and (if desired) the background model parameters:

$$\gamma_{ki} = \sum_{t,j,m} \tilde{\gamma}_{jkmi}(t) \tag{61}$$

$$\mathbf{m}_{ki} = \sum_{t,j,m} \tilde{\gamma}_{jkmi}(t)\mathbf{x}_{ki}(t) \tag{62}$$

$$\mathbf{S}_{ki} = \sum_{t,j,m} \tilde{\gamma}_{jkmi}(t)\mathbf{x}_{ki}(t)\mathbf{x}_{ki}(t)^T. \tag{63}$$

Note that $\mathbf{S}_{ki}$ is a symmetric quantity so we can store the lower triangular part. It is common to store the lower triangle of a matrix of size $N \times N$ as a vector of size $\frac{1}{2}N(N+1)$. The model mean information required for the within-class variance update can be derived from the weight statistics γ_{jkmi}, the model parameters and the statistics $\mathbf{Y}_{ki}$.

10.9. Statistics for speaker transforms

The statistics accumulation for constrained MLLR transformations is based on Equations (187) to (189) and can be written as follows:

$$\beta_k^{(s)} = \sum_{t\in\mathcal{T}(s),j,m,i} \tilde{\gamma}_{jkmi}(t) \tag{64}$$

$$\mathbf{K}_k^{(s)} = \sum_{t\in\mathcal{T}(s),j,m,i} \tilde{\gamma}_{jkmi}(t)\mathbf{\Sigma}_{ki}^{-1}\boldsymbol{\mu}_{jkmi}^{(s)}\mathbf{x}(t)^{+T} \tag{65}$$

$$\mathbf{S}_{ki}^{(s)} = \sum_{t\in\mathcal{T}(s),j,m} \tilde{\gamma}_{jkmi}(t)\mathbf{x}(t)^{+}\mathbf{x}(t)^{+T}, \tag{66}$$

where we need to compute:

$$\boldsymbol{\mu}_{jkmi}^{(s)} = \boldsymbol{\mu}_{jkmi} + \mathbf{o}_{ki}^{(s)}, \tag{67}$$

with $\boldsymbol{\mu}_{jkmi}$ as given in Equation (38) and $\mathbf{o}_{ki}^{(s)}$ as given in Equation (39).

11. UPDATES

In this section we describe the various kinds of parameter updates that can be done, along with a brief derivation for each. Derivation formulae are written in gray, so the reader who is only interested in implementation can easily skip them.

11.1. Model vectors update

Here we consider the update of the model vectors $\mathbf{v}_{jkm}$. Before doing the update we need to pre-compute the quantities:

$$\mathbf{H}_{ki} = \mathbf{M}_{ki}^T \boldsymbol{\Sigma}_{ki}^{-1} \mathbf{M}_{ki} \quad (68)$$

$$\gamma_{jkm} = \sum_i \gamma_{jkmi}. \quad (69)$$

Remembering that the effect of speaker transforms $\mathbf{W}_k^{(s)}$ and speaker projections and subspace $\mathbf{N}_{ki}$ and $\mathbf{v}_k^{(s)}$ are absorbed into the features $\mathbf{x}_{ki}(t)$, the part of the auxiliary function in $\mathbf{v}_{jkm}$ that relates to the means (not the weights) is:

$$\begin{aligned}
\mathcal{Q}_1(\mathbf{v}_{jkm}) &= K - 0.5 \sum_{t,i} \tilde{\gamma}_{jkmi}(t) \ldots \\
&\quad (\mathbf{x}_{ki}(t) - \boldsymbol{\mu}_{jkmi})^T \boldsymbol{\Sigma}_{ki}^{-1} (\mathbf{x}_{ki}(t) - \boldsymbol{\mu}_{jkmi}) \quad (70) \\
&= K' + \sum_{t,i} \tilde{\gamma}_{jkmi}(t) \boldsymbol{\mu}_{jkmi}^T \boldsymbol{\Sigma}_{ki}^{-1} \mathbf{x}_{ki}(t) \\
&\quad - 0.5 \sum_i \gamma_{jkmi} \boldsymbol{\mu}_{jkmi}^T \boldsymbol{\Sigma}_{ki}^{-1} \boldsymbol{\mu}_{jkmi} \quad (71) \\
&= K' + \sum_{t,i} \tilde{\gamma}_{jkmi}(t) {\mathbf{v}_{jkm}^+}^T \mathbf{M}_{ki}^T \boldsymbol{\Sigma}_{ki}^{-1} \mathbf{x}_{ki}(t) \\
&\quad - 0.5 \sum_i \gamma_{jkmi} {\mathbf{v}_{jkm}^+}^T \mathbf{M}_{ki}^T \boldsymbol{\Sigma}_{ki}^{-1} \mathbf{M}_{ki} \mathbf{v}_{jkm}^+ \quad (72) \\
&= K'' + \mathbf{v}_{jkm} \cdot \mathbf{y}_{jkm} \\
&\quad - 0.5 \sum_i \gamma_{jkmi} {\mathbf{v}_{jkm}^+}^T \mathbf{H}_{ki} \mathbf{v}_{jkm}^+ \quad (73) \\
&= K''' + \mathbf{v}_{jkm} \cdot (\mathbf{y}_{jkm} - \sum_i \gamma_{jkmi} \mathbf{h}_{ki(D+1)}^-) \\
&\quad - 0.5 \sum_i \gamma_{jkmi} \mathbf{v}_{jkm}^T \mathbf{H}_{ki}^{--} \mathbf{v}_{jkm}, \quad (74)
\end{aligned}$$

where $\mathbf{h}_{ki(D+1)}$ is the last row of $\mathbf{H}_{ki}$, $\mathbf{H}_{ki}^{--}$ is $\mathbf{H}_{ki}$ with the last row and column removed and $\mathbf{y}_{jkm}$ is as defined in Equation (53).

Now we consider a second part of the auxiliary function $\mathcal{Q}_2(\mathbf{v}_{jkm})$ which relates to the effect on the weights:

$$\begin{aligned}
\mathcal{Q}_2(\mathbf{v}_{jkm}) &= \sum_i \gamma_{jkmi} \log w_{jkmi} \quad (75) \\
&= \sum_i \gamma_{jkmi} \Big(\mathbf{w}_{ki} \cdot \mathbf{v}_{jkm}^+ - \\
&\qquad \log \textstyle\sum_{i'=1}^{I_k} \exp(\mathbf{w}_{ki'} \cdot \mathbf{v}_{jkm}^+) \Big) \quad (76)
\end{aligned}$$

Here we can use the inequality $1 - (x/\bar{x}) \leq -\log(x/\bar{x})$ (which is an equality at $x = \bar{x}$) to modify the auxiliary function as follows (note, x corresponds to the normalizing term $\sum_{i'=1}^{I_k} \exp(\mathbf{w}_{ki'} \cdot \mathbf{v}_{jkm}^+)$ and $\bar{x}$ to its current value):

$$\begin{aligned}
\mathcal{Q}_2'(\mathbf{v}_{jkm}) &= K + \sum_i \gamma_{jkmi} \Big(\mathbf{w}_{ki} \cdot \mathbf{v}_{jkm}^+ \\
&\qquad - \frac{\sum_{i'=1}^{I_k} \exp(\mathbf{w}_{ki'} \cdot \mathbf{v}_{jkm}^+)}{\sum_{i'=1}^{I_k} \exp(\mathbf{w}_{ki'} \cdot \bar{\mathbf{v}}_{jkm}^+)} \Big), \quad (77)
\end{aligned}$$

where $\bar{\mathbf{v}}_{jkm}$ is the current value of the speaker vector. Note that the denominator of the fraction is a constant. The motivation here is to simplify the calculus; the effect on convergence should be minimal as only one dimension of the problem is affected. Then we use a quadratic approximation to $\exp(x)$ around $x = x_0$, i.e. $\exp(x) \simeq \exp(x_0)(1 + (x - x_0) + 0.5(x - x_0)^2)$, and discard the terms constant in x to get $\exp(x) \simeq K + (x(1 - x_0) + 0.5x^2)\exp(x_0)$. This leads us to:

$$\begin{aligned}
\mathcal{Q}_2''(\mathbf{v}_{jkm}) &= K' + \textstyle\sum_i \gamma_{jkmi} \Big(\mathbf{w}_{ki} \cdot \mathbf{v}_{jkm}^+ - \quad (78) \\
&\qquad \frac{\sum_{i'=1}^{I_k} \left(\mathbf{w}_{ki'} \cdot \mathbf{v}_{jkm}^+ (1 - \mathbf{w}_{ki'} \cdot \bar{\mathbf{v}}_{jkm}^+) + 0.5 (\mathbf{w}_{ki'} \cdot \mathbf{v}_{jkm}^+)^2 \right) \exp(\mathbf{w}_{ki'} \cdot \bar{\mathbf{v}}_{jkm}^+)}{\sum_{i'=1}^{I_k} \exp(\mathbf{w}_{ki'} \cdot \bar{\mathbf{v}}_{jkm}^+)} \Big),
\end{aligned}$$

and we can simplify this using $\bar{w}_{jkmi} = \frac{\exp(\mathbf{w}_{ki} \cdot \bar{\mathbf{v}}_{jkm}^+)}{\sum_{i'} \exp(\mathbf{w}_{ki} \cdot \bar{\mathbf{v}}_{jkm}^+)}$ to:

$$\begin{aligned}
\mathcal{Q}_2''(\mathbf{v}_{jkm}) &= K' + \textstyle\sum_i \gamma_{jkmi} \quad \mathbf{w}_{ki} \cdot \mathbf{v}_{jkm}^+ \\
&\quad - \gamma_{jkm} \textstyle\sum_{i'=1}^{I_k} \bar{w}_{jkmi'} \quad \big(\mathbf{w}_{ki'} \cdot \mathbf{v}_{jkm}^+ (1 - \mathbf{w}_{ki'} \cdot \bar{\mathbf{v}}_{jkm}^+) \\
&\qquad + 0.5 (\mathbf{w}_{ki'} \cdot \mathbf{v}_{jkm}^+)^2 \big). \quad (79)
\end{aligned}$$

We can write it in terms of just the $\mathbf{v}_{jkm}$ (getting rid of the $\cdot^+$) as:

$$\begin{aligned}
\mathcal{Q}_2''(\mathbf{v}_{jkm}) &= K'' + \textstyle\sum_i \gamma_{jkmi} \quad \mathbf{w}_{ki}^- \cdot \mathbf{v}_{jkm} \\
&\quad - \gamma_{jkm} \textstyle\sum_{i'=1}^{I_k} \bar{w}_{jkmi} \big(\mathbf{w}_{ki'}^- \cdot \mathbf{v}_{jkm} (1 - \mathbf{w}_{ki'}^- \cdot \bar{\mathbf{v}}_{jkm}) \\
&\qquad + 0.5 \left(\mathbf{w}_{ki'}^- \cdot \mathbf{v}_{jkm} \right)^2 \big). \quad (80)
\end{aligned}$$

Certain extra terms appear here (if offsets are used) but are simplified out.

Note that at this point we have no guarantee that increasing the auxiliary function will increase the objective function, because the quadratic approximation to an exponential function is not a lower bound. We will just ignore this problem as it affects the estimation of the vectors $\mathbf{v}_{jkm}$, because we believe that the quadratic term will be dominated by the means rather than the weights in most cases. In the main situation where we expect an exception to this, i.e. when only one Gaussian has substantial nonzero counts for a particular sub-state, the smoothing process described in Section 11.1.1 should prevent divergence. We will deal with the question of convergence more rigorously when it is time to estimate the weight projections $\mathbf{w}_{ki}$.

Defining $\mathcal{Q}(\mathbf{v}_{jkm}) = \mathcal{Q}_1(\mathbf{v}_{jkm}) + \mathcal{Q}_2''(\mathbf{v}_{jkm})$ and gathering together the linear and quadratic terms in $\mathbf{v}_{jkm}^+$ from Equations (74)

and (80), we have:

$$\begin{aligned}
\mathcal{Q}(\mathbf{v}_{jkm}) &= K + \mathbf{v}_{jkm} \cdot \mathbf{g}_{jkm} - 0.5\mathbf{v}_{jkm}^T \mathbf{H}_{jkm} \mathbf{v}_{jkm} && (81)\\
\mathbf{g}_{jkm} &= \mathbf{y}_{jkm} - \sum_i \gamma_{jkmi} \bar{\mathbf{h}}_{ki(D+1)} && (82)\\
&\quad + \sum_i \bar{\hat{\mathbf{w}}}_{ki} \left(\gamma_{jkmi} - \gamma_{jkm} w_{jkmi} (1 - \bar{\hat{\mathbf{w}}}_{ki} \cdot \mathbf{v}_{jkm})\right)\\
\mathbf{H}_{jkm} &= \sum_i \gamma_{jkmi} \bar{\bar{\mathbf{H}}}_{ki}\\
&\quad + \gamma_{jkm} \sum_i \hat{w}_{jkmi} \bar{\hat{\mathbf{w}}}_{ki} \; \bar{\hat{\mathbf{w}}}_{ki}^T && (83)\\
\hat{\mathbf{v}}_{jkm} &= \mathbf{H}_{jkm}^{-1} \mathbf{g}_{jkm}. && (84)
\end{aligned}$$

Note that we use the notation $\hat{w}_{ki}$ and $\hat{w}_{jkmi}$ to mean that we should use the latest value if available, i.e. if the weight projections $\mathbf{w}_{ki}$ have been updated before the vectors $\mathbf{v}_{jkmi}$. Because $\mathbf{H}_{jkm}$ can have poor condition we should solve Equation (84) using the procedure described in Appendix G.

11.1.1. Smoothing

We routinely use a different approach that also handles the problem of singular matrices when estimating the parameters $\mathbf{v}_{jkm}$. This was originally developed as a workaround for the problem of singular matrices but is retained because it may improve the generalization of the model. It can be described in terms of a prior over the vectors $\mathbf{v}_{jkm}$ where the prior is not estimated from the data but is based on an *ad hoc* but dimensionally appropriate formula. In this sense it is similar to the Maximum A Posteriori (MAP) adaptation formulas based on a smoothing value τ that are used in the HTK toolkit [7]. We modify $\mathbf{H}_{jkm}$ and $\mathbf{g}_{jkm}$ as follows:

$$\begin{aligned}
\mathbf{H}'_{jkm} &= \mathbf{H}_{jkm} + \tau^{\text{vec}} \mathbf{H}_k^{(\text{sm})} && (85)\\
\mathbf{g}'_{jkm} &= \mathbf{g}_{jkm} + \tau^{\text{vec}} \mathbf{y}_k^{(\text{sm})} && (86)\\
\mathbf{H}_k^{(\text{sm})} &= \frac{1}{\sum_i \gamma_{ki}} \sum_i \gamma_{ki} \mathbf{H}_{ki} && (87)\\
\mathbf{y}_k^{(\text{sm})} &= \frac{1}{\sum_i \gamma_{ki}} \sum_{j,m} \mathbf{y}_{jkm}, && (88)\\
& && (89)
\end{aligned}$$

with $\gamma_{ki} = \sum_{j,m} \gamma_{jkmi}$, and to use $\mathbf{H}'_{jkm}$ and $\mathbf{g}'_{jkm}$ in place of $\mathbf{H}_{jkm}$ and $\mathbf{g}_{jkm}$ in Equation (84), for some chosen τ^{vec}, e.g. 20. This smoothing term ignores the effect of the vectors on the weights because we believe that is less important and to make the computation of the smoothing terms faster. For each sub-model k this smoothing formula equates to a prior centered around $\mathbf{H}_k^{(\text{sm})^{-1}} \mathbf{y}_k^{(\text{sm})}$ and with a variance equal to $\frac{1}{\tau^{\text{vec}}} \mathbf{H}_k^{(\text{sm})^{-1}}$. The center of prior is the same as the Maximum Likelihood estimate of the vectors for that sub-model if we forced them to all have the same value (and ignoring the effect on the weights).

We should still use the robust techniques described in Appendix G to solve the inversion problem, in case the modified $\mathbf{H}'_{jkm}$ is still poorly conditioned.

In Appendix C we will describe a more sophisticated alternative technique to handle the problem of over-training, based on estimating priors.

11.1.2. Auxiliary function improvement

We can test the auxiliary function improvement using Equation (81), measuring its difference in value before and after the update. Note that K just means an arbitrary constant, which we can ignore. The smoothed value of $\mathbf{H}'_{jkm}$ should not be substituted for $\mathbf{H}_{jkm}$ in Equation (81). It would be possible to get closer to the true likelihood improvement by using the exact auxiliary function for the weights (Equation (75)) rather than the quadratic approximation, but that is probably not necessary. Refer to Appendix C for a more principled solution to the estimation of the vectors, which is able to model correlations between different sub-models.

11.2. Sub-state weights estimation

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

$$\mathcal{Q}(c...) = \sum_{j,k,m} \gamma_{jkm} \log c_{jkm}, \qquad (90)$$

with the data count γ_{jkm} as defined in Equation (69). The sub-state weights must sum to 1 over all k, i for a particular j, so the Maximum Likelihood update is:

$$\hat{c}_{jkm} = \frac{\gamma_{jkm}}{\sum_{k=1}^{K} \sum_{m=1}^{M_{jk}} \gamma_{jkm}}. \qquad (91)$$

The objective function change can be computed by measuring Equation (90) before and after the update. A suitable smoothing approach to avoid getting zero weights (which might cause problems due to the interaction with pruning) is to define a value $\tau^{(w)}$, e.g. set to 5, and do:

$$\hat{c}_{jkm} = \frac{\gamma_{jkm} + \tau^{(w)}}{\sum_{k=1}^{K} \sum_{m=1}^{M_{jk}} \left(\gamma_{jkm} + \tau^{(w)}\right)}. \qquad (92)$$

11.3. Sub-state splitting

As in a normal mixture of Gaussians system, it is necessary to split the sub-states represented by the vectors $\mathbf{v}_{jkm}$ in order to eventually reach some target number of sub-states. For instance, we might have a target number of $T = 30000$ for a system with $J = 7000$ states. A robust way to assign the number of sub-states for each (state, sub-model) pair (j, k) is to have them proportional to some small power of the total data count γ_{jk} of the sub-model of the state, e.g. to the power $p = 0.2$ (e.g. as supported in HTK by the `PS` command in `HHEd` [7]), so we would have a target $T(j,k) = \max(1, \lfloor 0.5 + (T\gamma_{jk}^p / \sum_{j=1}^{J} \gamma_{jk}^p) \rfloor)$. The total number may differ somewhat from the target due to rounding. We would designate a subset of the iterations as iterations to split on, and would typically separate these by a few iterations to give the previously split vectors time to separate. On each iteration that we intend to split on, we would designate a target number T of sub-states that would gradually rise to the final desired number, and work out the state-specific targets $T(j,k)$ accordingly. Within a state j and sub-model k, after working out how many mixtures we have to split we would choose a subset of mixtures m to split based on highest count γ_{jkm}. The splitting should take place after other phases of update, to avoid the need to split the statistics. Let us suppose we are splitting vector $\mathbf{v}_{jkm}$ to the two vectors $\mathbf{v}_{jkm}$ and $\mathbf{v}_{jkm'}$, with m' a newly allocated mixture

index. We would split the weight and the vector as follows:

$$\hat{c}_{jkm} = \tfrac{1}{2} c_{jkm} \quad (93)$$
$$\hat{c}_{jkm'} = \tfrac{1}{2} c_{jkm} \quad (94)$$
$$\mathbf{r} = \text{Normally distributed random} \quad (95)$$
$$\hat{\mathbf{v}}_{jkm} = \mathbf{v}_{jkm} + 0.1 \mathbf{H}_k^{(\text{sm})\,-0.5} \mathbf{r} \quad (96)$$
$$\hat{\mathbf{v}}_{jkm'} = \mathbf{v}_{jkm} - 0.1 \mathbf{H}_k^{(\text{sm})\,-0.5} \mathbf{r}, \quad (97)$$

where we use $\mathbf{H}_k^{(\text{sm})}$ as defined in Equation (87). This formula is analogous to splitting a Gaussian mean by 0.1 standard deviations, e.g. as done in HTK [7]. $\mathbf{H}_k^{(\text{sm})}$ is analogous to a precision (inverse covariance matrix) so we get something like a standard deviation by taking it to the power -0.5.

11.4. Update for model projections

Now let us consider the auxiliary function for the model projection parameter $\mathbf{M}_{ki}$. It is similar to the one for the vectors in Equation (70):

$$\mathcal{Q}(\mathbf{M}_{ki}) = K - 0.5 \sum_{t,j,m} \tilde{\gamma}_{jkmi}(t) \ldots (\mathbf{x}_{ki}(t) - \boldsymbol{\mu}_{jkmi})^T \boldsymbol{\Sigma}_{ki}^{-1} (\mathbf{x}_{ki}(t) - \boldsymbol{\mu}_{jkmi}) \quad (98)$$
$$= K' + \sum_{t,j,m} \tilde{\gamma}_{jkmi}(t) \boldsymbol{\mu}_{jkmi}^T \boldsymbol{\Sigma}_{ki}^{-1} \mathbf{x}_{ki}(t) - 0.5 \sum_{j,m} \gamma_{jkmi} \boldsymbol{\mu}_{jkmi}^T \boldsymbol{\Sigma}_{ki}^{-1} \boldsymbol{\mu}_{jkmi} \quad (99)$$
$$= K' + \sum_{t,j,m} \tilde{\gamma}_{jkmi}(t) \mathbf{v}_{jkm}^{+\,T} \mathbf{M}_{ki}^T \boldsymbol{\Sigma}_{ki}^{-1} \mathbf{x}_{ki}(t) - 0.5 \sum_{j,m} \gamma_{jkmi} \mathbf{v}_{jkm}^{+\,T} \mathbf{M}_{ki}^T \boldsymbol{\Sigma}_{ki}^{-1} \mathbf{M}_{ki} \mathbf{v}_{jkm}^{+} \quad (100)$$

At this point we do the substitution $\mathbf{M}'_{ki} = \boldsymbol{\Sigma}_{ki}^{-0.5} \mathbf{M}_{ki}$. Then Equation (100) becomes:

$$\mathcal{Q}(\mathbf{M}_{ki}) = K' + \sum_{t,j,m} \tilde{\gamma}_{jkmi}(t) \mathbf{v}_{jkm}^{+\,T} \mathbf{M}'^{\,T}_{ki} \boldsymbol{\Sigma}_{ki}^{-0.5} \mathbf{x}_{ki}(t) - 0.5 \sum_{j,m} \gamma_{jkmi} \mathbf{v}_{jkm}^{+\,T} \mathbf{M}'^{\,T}_{ki} \mathbf{M}'_{ki} \mathbf{v}_{jkm}^{+} \quad (101)$$
$$= K' + \text{tr}\left(\mathbf{M}'^{\,T}_{ki} \boldsymbol{\Sigma}_{ki}^{-0.5} \mathbf{Y}_{ki}\right) - 0.5 \text{tr}\left(\mathbf{M}'_{ki} \mathbf{Q}_{ki} \mathbf{M}'^{\,T}_{ki}\right) \quad (102)$$

where

$$\mathbf{Y}_{ki} = \sum_{t,j,m} \tilde{\gamma}_{jkmi}(t) \mathbf{x}_{ki}(t) \mathbf{v}_{jkm}^{+\,T} \quad (103)$$
$$\mathbf{Q}_{ki} = \sum_{j,m} \gamma_{jkmi} \mathbf{v}_{jkm}^{+} \mathbf{v}_{jkm}^{+\,T}, \quad (104)$$

and note that $\mathbf{Y}_{ki}$ is part of the statistics (Equation (103) is the same as (54)) and $\mathbf{Q}_{ki}$ can be worked out from the count statistics and the model vectors. Now, $\mathbf{A}^T \mathbf{A}$ is the same as the same as $\sum_j \mathbf{a}_j \mathbf{a}_j^T$, where $\mathbf{a}_j$ are the rows of $\mathbf{A}$. So the auxiliary function of Equation (102) can be separated across the rows $\mathbf{m}'_{kid}$ of $\mathbf{M}'_{ki}$. Defining

$$\mathbf{Y}'_{ki} = \boldsymbol{\Sigma}_{ki}^{-0.5} \mathbf{Y}_{ki} \quad (105)$$

and $\mathbf{y}'_{kid}$ as the d'th row of $\mathbf{Y}'_{ki}$, we have:

$$\mathcal{Q}(\mathbf{m}'_{kid}) = \mathbf{m}'_{kid} \mathbf{y}'_{kid} - 0.5 \mathbf{m}'^{\,T}_{kid} \mathbf{Q}_{ki} \mathbf{m}'_{kid} \quad (106)$$
$$\hat{\mathbf{m}}'_{kid} = \mathbf{Q}_{ki}^{-1} \mathbf{y}'_{kid} \quad (107)$$
$$\mathbf{M}'_{ki} = \mathbf{Y}'_{ki} \mathbf{Q}_{ki}^{-1} \quad (108)$$
$$\hat{\mathbf{M}}_{ki} = \boldsymbol{\Sigma}_{ki}^{0.5} \hat{\mathbf{M}}'_{ki}. \quad (109)$$

so the update is:

$$\hat{\mathbf{M}}_{ki} = \mathbf{Y}_{ki} \mathbf{Q}_{ki}^{-1}. \quad (110)$$

If we are on the first iteration of model update then $\mathbf{Q}_{ki}$ will not be invertible and the update of the model projections should be skipped. To measure the auxiliary function change we can use:

$$\mathcal{Q}(\mathbf{M}_{ki}) = \text{tr}(\mathbf{M}_{ki}^T \boldsymbol{\Sigma}_{ki}^{-1} \mathbf{Y}_{ki}) - 0.5 \text{tr}(\boldsymbol{\Sigma}_{ki}^{-1} \mathbf{M}_{ki} \mathbf{Q}_{ki} \mathbf{M}_{ki}^T), \quad (111)$$

and measure the difference in this quantity before and after the update.

In Section J we describe an approach to estimate a prior over the matrices $\mathbf{M}_{ki}$ and estimate them on a Maximum A Posteriori basis; however, it is not clear that this is helpful.

11.5. Update for speaker projections

There is a symmetry between the model and speaker factors (except as regards the weights, which do not concern us here). Therefore the update for the speaker projections follows the same pattern as Section 11.4 above, except that the quadratic term corresponding to $\mathbf{Q}_{ki}$ above is now obtained during accumulation rather than from the model, since the speaker factors would typically not be stored with the model. This quantity we call $\mathbf{R}_{ki}$ and it is part of the accumulated statistics, see Equation (60). We also stored a linear term $\mathbf{Z}_{ki}$ in Equation (59). The update is:

$$\hat{\mathbf{N}}_{ki} = \mathbf{Z}_{ki} \mathbf{R}_{ki}^{-1}, \quad (112)$$

and the auxiliary function change can be measured by taking the difference before and after update, of:

$$\mathcal{Q}(\mathbf{N}_{ki}) = \text{tr}\left(\mathbf{N}_{ki}^T \boldsymbol{\Sigma}_{ki}^{-1} \mathbf{Z}_{ki}\right) - 0.5 \text{tr}\left(\boldsymbol{\Sigma}_{ki}^{-1} \mathbf{N}_{ki} \mathbf{R}_{ki} \mathbf{N}_{ki}^T\right). \quad (113)$$

11.6. Update for speaker vectors

The update for the speaker vectors $\mathbf{v}_k^{(s)}$ is a speaker-specific update. We use the statistics accumulated in Equations (57) and (58). It is analogous to the update for the model vectors, except without the extra terms relating to the weights. We skip the derivation because it is just a simpler form of the derivation in Section 11.1. Prior to estimating vectors for any speaker, we need to compute the quantities:

$$\mathbf{H}_{ki}^{\text{spk}} = \mathbf{N}_{ki}^T \boldsymbol{\Sigma}_{ki}^{-1} \mathbf{N}_{ki}, \quad (114)$$

which are speaker-subspace versions of (68). The auxiliary function and update rule are:

$$\mathcal{Q}(\mathbf{v}_k^{(s)}) = K + \mathbf{v}_k^{(s)} \cdot \mathbf{g}_k^{(s)} - 0.5 \mathbf{v}_k^{(s)\,T} \mathbf{H}_k^{(s)} \mathbf{v}_k^{(s)} \quad (115)$$
$$\mathbf{H}_k^{(s)} = \sum_i \gamma_{ki}^{(s)} \mathbf{H}_{ki}^{\text{spk}\,--} \quad (116)$$
$$\mathbf{g}_k^{(s)} = \mathbf{y}_k^{(s)} - \sum_i \gamma_{ki}^{(s)} \mathbf{h}_{ki(D+1)}^{(spk)\,--} \quad (117)$$
$$\hat{\mathbf{v}}_k^{(s)} = \mathbf{H}_k^{(s)\,-1} \mathbf{g}_k^{(s)}. \quad (118)$$

This update is sufficient when there is enough data to estimate each vector $\mathbf{v}_k^{(s)}$. However, it could lead to over-training otherwise, especially for large K. It would be nice to be able to estimate some kind of prior over the whole set of vectors $\mathbf{v}_k^{(s)}$, but this would require some effort because K might be quite large so the dimension of the concatenated vector would be high, and also because we would have to estimate the prior from estimates of the vectors. Instead, for now we will opt for a much simpler approach that uses a τ value to smooth each $\mathbf{v}_k^{(s)}$ back to a global $\mathbf{v}^{(s)}$ (shared for all k). First we estimate a globally shared value of $\mathbf{v}^{(s)}$:

$$\mathbf{H}^{(s)} = \sum_k \mathbf{H}_k^{(s)} \quad (119)$$

$$\mathbf{g}^{(s)} = \sum_k \mathbf{g}_k^{(s)} \quad (120)$$

$$\hat{\mathbf{v}}^{(s)} \leftarrow \mathbf{H}^{(s)-1}\mathbf{g}^{(s)}. \quad (121)$$

Then we interpolate between the global and sub-model specific version of the vector, as follows.

$$\gamma_k^{(s)} = \sum_i \gamma_{ki}^{(s)} \quad (122)$$

$$\hat{\mathbf{v}'}_k^{(s)} = \frac{\gamma_{ki}}{\tau^{\text{spk}} + \gamma_k^{(s)}}\hat{\mathbf{v}}_k^{(s)} + \frac{\tau^{\text{spk}}}{\tau^{\text{spk}} + \gamma_k^{(s)}}\hat{\mathbf{v}}^{(s)} \quad (123)$$

The value τ^{spk} may be interpreted as a number of frames; we suggest a value of $\tau^{\text{spk}} = T$ (the same as the speaker subspace dimension) but it is worth experimenting with. Note that this technique assumes that the speaker subspaces for the different values of k are related in some sensible way. We can ensure this by setting τ^{spk} to a very large value (e.g. 1.0e+10) for the first few iterations of training, say the first ten iterations, and thereafter leaving it at least as large as the value to be used in testing. To compute the change in auxiliary function, we can work out the change in Equation (115) before and after update.

11.7. Update for weight projections

The weight projections $\mathbf{w}_{ki}$ are updated using an approach similar to the one used in Section 11.1 for the model vectors. The auxiliary function for the set of weight projection vectors $\mathbf{w}_{k1} \dots \mathbf{w}_{kI_k}$ for sub-model k is as follows (using $\mathbf{w}_{k\cdot}$ to refer to this whole set of vectors):

$$\begin{aligned}\mathcal{Q}(\mathbf{w}_{k\cdot}) &= \sum_{j,m,i} \gamma_{jkmi} \log w_{jkmi} \\ &= \sum_{j,m,i} \gamma_{jkmi} \left(\mathbf{w}_{ki} \cdot \mathbf{v}_{jkm}^{+} - \log \sum_{i'=1}^{I_k} \exp(\mathbf{w}_{ki'} \cdot \mathbf{v}_{jkm}^{+}) \right). \quad (124)\end{aligned}$$

Following similar steps to Section 11.1 using the inequality $1 - (x/\bar{x}) \leq -\log(x/\bar{x})$, where $\bar{x}$ is the current value of x, we arrive at:

$$\mathcal{Q}'(\mathbf{w}_{k\cdot}) = K + \sum_{j,m,i} \gamma_{jkmi} \left(\mathbf{w}_{ki} \cdot \mathbf{v}_{jkm}^{+} - \frac{\sum_{i'=1}^{I_k} \exp(\mathbf{w}_{ki'} \cdot \mathbf{v}_{jkm}^{+})}{\sum_{i'=1}^{I_k} \exp(\bar{\mathbf{w}}_{ki'} \cdot \mathbf{v}_{jkm}^{+})} \right), \quad (125)$$

where $\bar{\mathbf{w}}_{ki}$ is the current value of the weight projection vector, which is equivalent to:

$$\mathcal{Q}'(\mathbf{w}_{k\cdot}) = K' + \sum_{j,m,i} \gamma_{jkmi} \left(\mathbf{w}_{ki} \cdot \mathbf{v}_{jkm}^{+} - \sum_{j,m} \gamma_{jkm} \frac{\exp(\mathbf{w}_{ki} \cdot \mathbf{v}_{jkm}^{+})}{\sum_{i'=1}^{I_k} \exp(\bar{\mathbf{w}}_{ki'} \cdot \mathbf{v}_{jkm}^{+})} \right), \quad (126)$$

with $\gamma_{jkm} = \sum_i \gamma_{jkmi}$. Note that at this point the auxiliary function can be separated into terms for each i, with $\mathcal{Q}'(\mathbf{w}_{k\cdot}) = K' + \sum_i \mathcal{Q}'(\mathbf{w}_{ki})$, and:

$$\mathcal{Q}'(\mathbf{w}_{ki}) = \sum_{j,m} \gamma_{jkmi} \left(\mathbf{w}_{ki} \cdot \mathbf{v}_{jkm}^{+} - \sum_{j,m} \gamma_{jkm} - \frac{\exp(\mathbf{w}_{ki} \cdot \mathbf{v}_{jkm}^{+})}{\sum_{i'=1}^{I_k} \exp(\bar{\mathbf{w}}_{ki'} \cdot \mathbf{v}_{jkm}^{+})} \right). \quad (127)$$

We can compute the first and second derivatives of this with respect to the vector $\mathbf{w}_{ki}$ as follows:

$$\frac{\partial \mathcal{Q}'(\mathbf{w}_{ki})}{\partial \mathbf{w}_{ki}}^T = \sum_{j,m} (\gamma_{jkmi} - \gamma_{jkm} w_{jkmi}) \mathbf{v}_{jkm}^{+} \quad (128)$$

$$\frac{\partial^2 \mathcal{Q}'(\mathbf{w}_{ki})}{\partial \mathbf{w}_{ki}^2} = -\sum_{j,m} \gamma_{jkm} w_{jkmi} \mathbf{v}_{jkm}^{+} \mathbf{v}_{jkm}^{+}{}^T, \quad (129)$$

using $w_{jkmi} = \frac{\exp(\mathbf{w}_{ki} \cdot \mathbf{v}_{jkm}^{+})}{\exp(\sum_{i'} \mathbf{w}_{ki'} \cdot \mathbf{v}_{jkm}^{+})}$. A natural approach would be to take

$$\hat{\mathbf{w}}_{ki} = \mathbf{w}_{ki} - \left(\frac{\partial^2 \mathcal{Q}'(\mathbf{w}_{ki})}{\partial \mathbf{w}_{ki}^2} \right)^{-1} \frac{\partial \mathcal{Q}'(\mathbf{w}_{ki})}{\partial \mathbf{w}_{ki}}^T, \quad (130)$$

to do this for all i simultaneously and to measure the auxiliary function of Equation (126); if the auxiliary function increased we would accept the update and otherwise we would keep halving the step size until the auxiliary function increased. Unfortunately this approach does not seem to be stable and it often becomes necessary to halve the step size many times. The reason is that a two-term Taylor series approximation is a very poor approximation to the exponential function if we are moving too far. We find the convergence is better with the following approximation which amounts to making the quadratic part of the approximation larger (more negative) in certain cases. We do this by replacing the term $\gamma_{jkm} w_{jkmi}$ in Equation (129) with $\max(\gamma_{jkmi}, \gamma_{jkm} w_{jkmi})$. The reason is that the Maximum Likelihood solution for the weight w_{jkmi} without the subspace constraint would be $\gamma_{jkmi}/\gamma_{jkm}$, and we believe that the update should take us closer to the Maximum Likelihood estimate. At that point $\gamma_{jkm} w_{jkmi}$ would take on the value γ_{jkmi}. We take the larger of the two for safety. This leads to the following auxiliary function and update.

$$\mathcal{Q}''(\mathbf{w}_{ki}) = \mathbf{w}_{ki} \cdot \mathbf{g}_{ki} - \frac{1}{2} \mathbf{w}_{ki}^T \mathbf{F}_{ki} \mathbf{w}_{ki} \quad (131)$$

$$\mathbf{g}_{ki} = \sum_{j,m} (\gamma_{jkmi} - \gamma_{jkm} w_{jkmi}) \mathbf{v}_{jkm}^{+} \quad (132)$$

$$\mathbf{F}_{ki} = \sum_{j,m} \max(\gamma_{jkmi}, \gamma_{jkm} w_{jkmi}) \mathbf{v}_{jkm}^{+} \mathbf{v}_{jkm}^{+}{}^T \quad (133)$$

$$\hat{\mathbf{w}}_{ki} = \mathbf{w}_{ki} + \mathbf{F}_{ki}^{-1} \mathbf{g}_{ki}. \quad (134)$$

Because $\mathbf{F}_{ki}$ can be of reduced rank or have poor condition, we should use the techniques described in Appendix G to solve this problem. Note that we are solving for a change Δ_{ki} in $\mathbf{w}_{ki}$ where the auxiliary function is $\Delta_{ki} \cdot \mathbf{g}_{ki} - \frac{1}{2}\Delta_{ki}^T \mathbf{F}_{ki} \Delta_{ki}$ adn the initial value of Δ_{ki} is zero.

After doing the update of Equation (134) for all i, we should check the auxiliary function value of Equation (126), and if it has not increased keep halving the step size until the auxiliary function change is positive. We have never observed the halving of step size to take place, though. The whole procedure can then be repeated for, say, three iterations, i.e. do Equation (132) through (134) for all i, set $\mathbf{w}_{ki} := \hat{\mathbf{w}}_{ki}$ for all i and repeat. Note that if we are updating the vectors $\mathbf{v}_{jkm}$ before the weight projections it is the updated vectors $\hat{\mathbf{v}}_{jkm}$ that should appear in Equations (132) and (133), and likewise as mentioned in Section 11.1 if we update $\mathbf{w}_{ki}$ first we should use their updated values $\hat{\mathbf{w}}_{ki}$ during the update of $\mathbf{v}_{jkmi}$.

The change in the approximated quadratic auxiliary function of Equation (131) and the change in the exact auxiliary function of Equation (126) should be measured as diagnostics on each iteration of weight update. The two auxiliary functions should both increase on each iteration and if the approximation is good they should both increase by a similar amount.

11.8. Update for within-class variances

The auxiliary function for the within-class variances $\mathbf{\Sigma}_{ki}$ can be written as follows:

$$\mathcal{Q}(\mathbf{\Sigma}_{ki}) = K - 0.5 \sum_{j,m,t} \tilde{\gamma}_{jkmi}(t) \Big(\log\det \mathbf{\Sigma}_{ki} \tag{135}$$

$$+ (\mathbf{x}_{ki}(t) - \boldsymbol{\mu}_{jkmi})^T \mathbf{\Sigma}_{ki}^{-1} (\mathbf{x}_{ki}(t) - \boldsymbol{\mu}_{jkmi}) \Big). \tag{136}$$

Without doing the derivation as this type of update is very common, the answer is:

$$\hat{\mathbf{\Sigma}}_{ki} = \frac{\sum_{j,m,t} \tilde{\gamma}_{jkmi}(t) (\mathbf{x}_{ki}(t) - \boldsymbol{\mu}_{jkmi})(\mathbf{x}_{ki}(t) - \boldsymbol{\mu}_{jkmi})^T}{\sum_{j,m,t} \tilde{\gamma}_{jkmi}(t)}$$

$$= \frac{1}{\gamma_{ki}} \Bigg(\mathbf{S}_{ki} - \sum_{j,m,t} \tilde{\gamma}_{jkmi}(t) \boldsymbol{\mu}_{jkmi} \mathbf{x}_{ki}(t)^T$$

$$- \sum_{j,m,t} \tilde{\gamma}_{jkmi}(t) \mathbf{x}_{ki}(t) \boldsymbol{\mu}_{jkmi}^T + \sum_{j,m} \gamma_{jkmi} \boldsymbol{\mu}_{jkmi} \boldsymbol{\mu}_{jkmi}^T \Bigg) \tag{137}$$

$$\mathbf{S}_{ki} = \sum_{j,m,t} \tilde{\gamma}_{jkmi}(t) \mathbf{x}_{ki}(t) \mathbf{x}_{ki}(t)^T \tag{138}$$

$$\gamma_{ki} = \sum_{t,j,m} \tilde{\gamma}_{jkmi}(t) \tag{139}$$

$$= \sum_{j,m} \gamma_{jkmi} \tag{140}$$

In Equation (137), the outer product of the data itself and the corresponding counts have been accumulated (we accumulated $\mathbf{S}_{ki}$ and γ_{ki}; Equation (138) is the same as (63) and Equation (139) is the same as (61)); now we show how we can compute the cross terms between the data and the means from statistics $\mathbf{Y}_{ki}$ which were accumulated in order to update the projections $\mathbf{M}_{ki}$. Recalling that

$$\mathbf{Y}_{ki} = \sum_{t,j,m} \tilde{\gamma}_{jkmi}(t) \mathbf{x}_{ki}(t) \mathbf{v}_{jkm}^{+\,T} \tag{141}$$

which is a repetition of Equation (54), we can right-multiply by $\mathbf{M}_{ki}^T$ to get:

$$\mathbf{Y}_{ki} \mathbf{M}_{ki}^T = \sum_{t,j,m} \tilde{\gamma}_{jkmi}(t) \mathbf{x}_{ki}(t) \boldsymbol{\mu}_{jkm}^T, \tag{142}$$

which is one of the cross terms we need in Equation (137) (we can transpose to get the other). As for the weighted outer product of the means, we can compute this as:

$$\mathbf{S}_{ki}^{\text{means}} = \sum_{j,m} \gamma_{jkmi} \boldsymbol{\mu}_{jkmi} \boldsymbol{\mu}_{jkmi}^T, \tag{143}$$

using $\boldsymbol{\mu}_{jkmi} = \mathbf{M}_{ki} \mathbf{v}_{jkm}^{+}$ to compute the means. The update equation is:

$$\hat{\mathbf{\Sigma}}_{ki} = \frac{1}{\gamma_{ki}} \Big(\mathbf{S}_{ki} + \mathbf{S}_{ki}^{\text{means}} - \mathbf{Y}_{ki} \mathbf{M}_{ki}^T - \mathbf{M}_{ki} \mathbf{Y}_{ki}^T \Big). \tag{144}$$

In order to handle the problem of very small counts and other reasons why $\hat{\mathbf{\Sigma}}_{ki}$ may be singular, it is desirable to floor $\hat{\mathbf{\Sigma}}_{ki}$ to a small fraction (e.g. $1/10$) of a global average variance quantity. This will make it unnecessary to enforce a minimum count. See Appendix I for a method of flooring full covariance matrices.

The auxiliary function improvement can be calculated as:

$$\Delta\mathcal{Q}(\mathbf{\Sigma}_{ki}) = -\frac{\gamma_{ki}}{2} \Big(\log\det \hat{\mathbf{\Sigma}}_{ki} - \log\det \mathbf{\Sigma}_{ki} + D - \mathrm{tr}\,(\mathbf{\Sigma}_{ki}^{-1} \hat{\mathbf{\Sigma}}_{ki}) \Big). \tag{145}$$

This is not valid in the presence of flooring but should be good enough for diagnostics.

11.9. Updating the "background" model

Here we describe the equations used to update the "background" GMM during training of the entire HMM set. This refers to any training of the "background" GMM parameters that is done while training the main model, not the "pre-training" which was done through standard E-M and alluded to in Section 7.2. There is actually no theoretical justification for training the "background" model during model training, since formally the background model parameters are not part of the model at all; they are only used for pruning. In fact, proofs of convergence the update formulae for the main model in the presence of pruning would only work if we left the background model constant. Despite this, there is a practical and intuitive reason why we might want to train the background model, which is to keep it in correspondence with the Gaussian posteriors of the main model, so the pruning can be more accurate. Experiments have failed to show any difference between updating and not updating the background model. We show the equations here anyway.

11.10. Updating the full-covariance background model

The full covariance background model can be updated (if desired) as follows:

$$\hat{\tilde{w}}_{ki} = \frac{\gamma_{ki}}{\sum_{k,i} \gamma_{ki}} \tag{146}$$

$$\hat{\tilde{\boldsymbol{\mu}}}_{ki} = \frac{1}{\gamma_{ki}} \mathbf{m}_{ki} \tag{147}$$

$$\hat{\tilde{\mathbf{\Sigma}}}_{ki} = \frac{1}{\gamma_{ki}} \mathbf{S}_{ki} - \hat{\tilde{\boldsymbol{\mu}}}_{ki} \hat{\tilde{\boldsymbol{\mu}}}_{ki}^T, \tag{148}$$

with γ_{ki}, $\mathbf{m}_{ki}$ and $\mathbf{S}_{ki}$ as accumulated in Equations (61) to (63). We can skip the update for a particular Gaussian if the count is very

small, e.g. less than D. Note that in practice we may just set all the weights to be all the same rather than using Equation (146). The variance should be floored to e.g. one tenth of a globally averaged variance, using the method described in Appendix I. The auxiliary function improvement from the mean and variance update is:

$$\begin{aligned}\Delta\mathcal{Q}(\bar{\boldsymbol{\mu}}_{ki}, \bar{\boldsymbol{\Sigma}}_{ki}) &= -\frac{\gamma_{ki}}{2}\Big(\log\det\hat{\bar{\boldsymbol{\Sigma}}}_{ki} - \log\det\bar{\boldsymbol{\Sigma}}_{ki} + D \\ &\quad -(\hat{\bar{\boldsymbol{\mu}}}_{ki} - \bar{\boldsymbol{\mu}}_{ki})^T\bar{\boldsymbol{\Sigma}}_{ki}^{-1}(\hat{\bar{\boldsymbol{\mu}}}_{ki} - \bar{\boldsymbol{\mu}}_{ki}) \\ &\quad -\mathrm{tr}\,(\bar{\boldsymbol{\Sigma}}_{ki}^{-1}\hat{\bar{\boldsymbol{\Sigma}}}_{ki})\Big). \end{aligned} \quad (149)$$

This is the improvement in the auxiliary function we use to update the full-covariance background model, which is not actually an auxiliary function for our overall data likelihood; it is simply useful to tell how much the background model is changing.

11.11. Updating the diagonal background model

Assuming the diagonal version of the background model is evaluated with features that have the same level of adaptation as the full version, we can just set its parameters to be the diagonalized version of the full background model's updated parameters:

$$\hat{\bar{\boldsymbol{\mu}}}_{ki}^{\mathrm{diag}} = \hat{\bar{\boldsymbol{\mu}}}_{ki} \quad (150)$$

$$\hat{\bar{\boldsymbol{\Sigma}}}_{ki}^{\mathrm{diag}} = \mathrm{diag}\,(\hat{\bar{\boldsymbol{\Sigma}}}_{ki}), \quad (151)$$

$$(152)$$

where $\mathrm{diag}\,(\mathbf{M})$ is $\mathbf{M}$ with all its off-diagonal elements set to zero.

12. ESTIMATING CONSTRAINED MLLR FOR THE FULL COVARIANCE CASE

Here we describe a method of estimating constrained MLLR transforms on a set of full-covariance Gaussians. We describe a method that is designed to be easily combinable with subspace techniques in which we constrain the transform to vary in a subspace of the full parameter space. The gist of the technique is that we compute an approximation to the Hessian of the likelihood function with respect to the transform matrix parameters, for data generated from the model itself. This tells us what the second gradient will be, approximately, for "typical" statistics. We can use this information to pre-scale the parameter space so that the expected second gradient is proportional to the unit matrix. Then when presented with actual data, we compute the sufficient statistics to update the transform, and the update then consists of repeatedly choosing the optimal step size in the direction of the gradient, within the pre-scaled parameter space. The pre-scaling ensures that this type of update will converge reasonably fast. The actual Hessian given the statistics we accumulated may differ somewhat from the one we pre-computed, but all we need is for the estimate to be in the right ballpark– a factor of two error, for instance, will not slow down the update too much. Because the core of this technique is a simple gradient descent method, it is easy to limit to a subspace of the matrix parameters; this is not the case for traditional row-by-row updates, which although they can be combined with subspace techniques [8] and full covariance models [9, 10], are hard to use efficiently with a combination of the two.

12.1. Constrained MLLR

The transformation we use in constrained MLLR is:

$$\mathbf{x} \rightarrow \mathbf{W}^{(s)}\mathbf{x}^{+}, \quad (153)$$

where $\mathbf{x}^{+}$ is $\mathbf{x}$ with a 1 appended to it, and the speaker transformation matrix $\mathbf{W}^{(s)}$ is a D by $D+1$ matrix which can be written as:

$$\mathbf{W}^{(s)} = \left[\mathbf{A}^{(s)}; \mathbf{b}^{(s)}\right], \quad (154)$$

where $\mathbf{A}^{(s)}$ is a square matrix and $\mathbf{b}^{(s)}$ is the offset term.

The objective function is the likelihood of the transformed data given the GMM, plus the log determinant of $\mathbf{A}^{(s)}$. The need for the log determinant term is clearer if we view this form of adaptation as a transformation of the model rather than the features [11], but the process is probably easier to visualize if we view it as a feature transformation.

12.2. Pre-transform

In order to make the second gradient computation easier, we first (conceptually) pre-transform the features and the model such that the average within-class variance is unit, the average mean is zero and the covariance of the mean vectors is diagonal. We will not have to apply this transformation to the model or the features, but the transforms we compute here will appear in the optimization formulae for the transforms we are estimating.

The input to this stage assumes we have Gaussian indices $1 \leq j \leq J$, with means $\boldsymbol{\mu}_j$ and (possibly full) variances $\boldsymbol{\Sigma}_j$, and occupation probabilities w_j such that $\sum_{j=1}^{J} w_j = 1$. By using this notation we do not assume that we have a flat mixture of Gaussians, it could be a HMM but in that case we have to work out the expected occupation probabilities w_j of the individual Gaussians within the whole HMM. It is not absolutely critical that these be exact; making approximations such as assuming that all states are equally likely would probably not affect the optimization speed too much.

We first compute:

$$\boldsymbol{\Sigma}_W = \sum_{j=1}^{J} w_j\boldsymbol{\Sigma}_j \quad (155)$$

$$\boldsymbol{\mu} = \sum_{j=1}^{J} w_j\boldsymbol{\mu}_j \quad (156)$$

$$\boldsymbol{\Sigma}_B = \left(\sum_{j=1}^{J} w_j\boldsymbol{\mu}_j\boldsymbol{\mu}_j^T\right) - \boldsymbol{\mu}\boldsymbol{\mu}^T. \quad (157)$$

We are computing a pre-transform $\mathbf{W}_{\mathrm{pre}} = [\mathbf{A}_{\mathrm{pre}}; \mathbf{b}_{\mathrm{pre}}]$, which will give our model the desired properties. The square part of the matrix $\mathbf{A}_{\mathrm{pre}}$ should be such that $\mathbf{A}_{\mathrm{pre}}\boldsymbol{\Sigma}_W\mathbf{A}_{\mathrm{pre}}^T = \mathbf{I}$ and $\mathbf{A}_{\mathrm{pre}}\boldsymbol{\Sigma}_B\mathbf{A}_{\mathrm{pre}}^T$ is diagonal, and we need $\mathbf{A}_{\mathrm{pre}}\boldsymbol{\mu} + \mathbf{b} = \mathbf{0}$. We first do the Cholesky decomposition

$$\boldsymbol{\Sigma}_W = \mathbf{L}\mathbf{L}^T, \quad (158)$$

compute $\mathbf{B} = \mathbf{L}^{-1}\boldsymbol{\Sigma}_B\mathbf{L}^{-T}$, do the singular value decomposition

$$\mathbf{B} = \mathbf{U}\mathbf{D}\mathbf{V}^T \quad (159)$$

(this implies $\mathbf{B} = \mathbf{U}\mathbf{D}\mathbf{U}^T$ because $\mathbf{B}$ is positive semi-definite, see Appendix B), and the transform we want is

$$\mathbf{A}_{\mathrm{pre}} = \mathbf{U}^T\mathbf{L}^{-1} \quad (160)$$

$$\boldsymbol{\mu}_{\mathrm{pre}} = -\mathbf{A}_{\mathrm{pre}}\boldsymbol{\mu} \quad (161)$$

$$\mathbf{W}_{\mathrm{pre}} = [\mathbf{A}_{\mathrm{pre}}; \mathbf{b}_{\mathrm{pre}}]. \quad (162)$$

We also need to compute the inverse transformation to $\mathbf{W}_{\mathrm{pre}}$, which we call $\mathbf{W}_{\mathrm{inv}}$ (this is not the same as $\mathbf{W}_{\mathrm{pre}}^{-1}$ as it is not square).

$$\mathbf{W}_{\mathrm{inv}} = \mathbf{W}_{\mathrm{pre}}^{+\,-1-} \quad (163)$$

$$= \left[\mathbf{A}_{\mathrm{pre}}^{-1}; \boldsymbol{\mu}\right]. \quad (164)$$

The notation $\cdot^+$ in this context means appending a row whose last element is 1 and the rest are zero; $\cdot^-$ means removing the last row.

12.3. Hessian computation

Now we compute the Hessian (matrix of second derivatives) of the expected per-frame model likelihood with respect to transform parameters $\mathbf{W}$ around the point where $\mathbf{W} = [\mathbf{I}; \mathbf{0}]$, for typical data generated from the model. At this point we assume the model has been transformed as above so the average within-class variance is unit, the scatter of the means equals $\mathbf{D}$, and the average mean is $\mathbf{0}$. In addition we are making the approximation that all the variances are equal to the average variance $\mathbf{I}$. Thus, we are in fact computing an approximated, expected Hessian. This approximation is not a problem since we are only using the computed Hessian for preconditioning the problem; it will not lead to any inaccuracy in the answer but only a slower rate of convergence.

The auxiliary function is:

$$\begin{aligned}\mathcal{Q}(\mathbf{W}) &= \log|\det \mathbf{A}| \\ &-0.5\sum_{j=1}^{J} w_j \mathcal{E}_j\left((\mathbf{A}\mathbf{x}+\mathbf{b}-\boldsymbol{\mu}_j)^T(\mathbf{A}\mathbf{x}+\mathbf{b}-\boldsymbol{\mu}_j)\right), \quad (165)\end{aligned}$$

where the expectation $\mathcal{E}_j$ is over typical features $\mathbf{x}$ generated from Gaussian j. The auxiliary function has a simple form because the variances $\boldsymbol{\Sigma}_j$ are assumed to be unit. Then we use the fact that the features $\mathbf{x}$ for Gaussian j are distributed with unit variance and mean $\boldsymbol{\mu}_j$, to get (keeping only terms quadratic in $\mathbf{A}$ and/or $\mathbf{b}$):

$$\begin{aligned}\mathcal{Q}(\mathbf{W}) &= K + \log|\det \mathbf{A}| \\ &-0.5\sum_{j=1}^{J} w_j \Big(\mathrm{tr}\left(\mathbf{A}(\mathbf{I}+\boldsymbol{\mu}_j\boldsymbol{\mu}_j^T)\mathbf{A}^T\right) \\ &+\mathbf{b}^T\mathbf{b} + 2(\mathbf{A}\boldsymbol{\mu}_j)\cdot\mathbf{b}\Big). \quad (166)\end{aligned}$$

Now we can use the fact that the means $\boldsymbol{\mu}_j$ have zero mean and variance $\mathbf{D}$, to get:

$$\begin{aligned}\mathcal{Q}(\mathbf{W}) &= K + \log|\det \mathbf{A}| \\ &-0.5\left(\mathrm{tr}\left(\mathbf{A}(\mathbf{I}+\mathbf{D})\mathbf{A}^T\right) + \mathbf{b}^T\mathbf{b}\right). \quad (167)\end{aligned}$$

We can work out the quadratic terms in the expansion of $\log|\det \mathbf{A}|$. If we use the fact that $\frac{\partial \log\det \mathbf{A}}{\partial \mathbf{A}} = \mathbf{A}^{-1}$ (using the convention where $(\frac{\partial f}{\partial \mathbf{A}})_{ij} = \frac{\partial f}{\partial a_{ji}}$), we have $\frac{\partial \log|\det \mathbf{A}|}{\partial a_{ij}} = (\mathbf{A}^{-1})_{ji}$. To find the second derivative we can use the fact that if the matrix $\mathbf{A}$ depends on a parameter t, $\frac{\mathbf{d}\mathbf{A}^{-1}}{\mathbf{d}t} = -\mathbf{A}^{-1}\frac{\mathbf{d}\mathbf{A}}{\mathbf{d}t}\mathbf{A}^{-1}$. Considering the special dependency where $a_{ij} = t$ and the rest are fixed, $\frac{\mathbf{d}\mathbf{A}}{\mathbf{d}t}$ would just equal $\mathbf{S}_{ij}$ which we define as matrix with a 1 in position i, j and zeros everywhere else. Evaluated as a constant around $\mathbf{A} = \mathbf{I}$, we have $\frac{\partial \mathbf{A}^{-1}}{\partial a_{ij}} = -\mathbf{A}^{-1}\mathbf{S}_{ij}\mathbf{A}^{-1} = -\mathbf{S}_{ij}$. This implies that $\frac{\partial (\mathbf{A}^{-1})_{ij}}{\partial a_{kl}} = -\delta(i,k)\delta(j,l)$. Thus we have:

$$\begin{aligned}\frac{\partial \log|\det \mathbf{A}|}{\partial a_{ij}a_{kl}} &= \frac{\partial (\mathbf{A}^{-1})_{ji}}{\partial a_{kl}} \quad (168)\\ &= -\delta(j,k)\delta(i,l) \quad (169)\end{aligned}$$

This means that the quadratic term in the Taylor expansion of $\log\det \mathbf{A}$ can be expressed as $-0.5\sum_{ij} a_{ij}a_{ji}$. Using this and doing a similar element-by-element expansion of the other terms in (167), using $\mathcal{Q}^{(2)}(\mathbf{W})$ to mean $\mathcal{Q}(\mathbf{W})$ with just the quadratic terms kept we can write:

$$\begin{aligned}\mathcal{Q}^{(2)}(\mathbf{W}) &= -0.5\sum_{i,j} a_{ij}a_{ji} + a_{ij}^2(1+d_j) \\ &\quad -0.5\sum_i b_i^2, \quad (170)\end{aligned}$$

where we use d_j for the j'th diagonal element of $\mathbf{D}$. Thus, the Hessian of the objective function in the elements of $\mathbf{W}$ has a particularly simple covariance structure where there is a correlation only between an element and its transpose. The diagonal of $\mathbf{A}$ and the elements of $\mathbf{b}$ are not correlated with anything. It would be possible to rearrange the elements of $\mathbf{A}$ and $\mathbf{b}$ into a big vector such that the Hessian had a block diagonal structure with blocks of size 2, but it will be easier to avoid that because the mapping would be somewhat complicated. Instead we write down the transformations that make the quadratic term equal to $-0.5\,\mathbf{I}$, as linear operations on the elements of $\mathbf{A}$.

Let us consider a vector $\mathbf{v} = \begin{bmatrix} a_{ij} \\ a_{ji} \end{bmatrix}$, where $i \neq j$, and let us arbitrarily stipulate that $j < i$ to fix the ordering. Writing down a matrix $\mathbf{M}$ such that the relevant terms in (170) would equal $-0.5\mathbf{v}^T\mathbf{M}\mathbf{v}$, we have:

$$\mathbf{M} = \begin{bmatrix} 1+d_j & 1 \\ 1 & 1+d_i \end{bmatrix}. \quad (171)$$

Working out the Cholesky decomposition of $\mathbf{M}$, we have:

$$\begin{aligned}\mathbf{M} &= \mathbf{L}\mathbf{L}^T \quad (172)\\ \mathbf{L} &= \begin{bmatrix} (1+d_j)^{0.5} & 0 \\ (1+d_j)^{-0.5} & \left(1+d_i-(1+d_j)^{-1}\right)^{0.5} \end{bmatrix} \quad (173)\end{aligned}$$

The inverse of $\mathbf{L}$ is, using $\begin{bmatrix} a & 0 \\ b & c \end{bmatrix}^{-1} = \begin{bmatrix} 1/a & 0 \\ -b/(ac) & 1/c \end{bmatrix}$:

$$\mathbf{L}^{-1} = \begin{bmatrix} (1+d_j)^{-0.5} & 0 \\ -\frac{\left(1+d_i-(1+d_j)^{-1}\right)^{-0.5}}{(1+d_j)} & \left(1+d_i-(1+d_j)^{-1}\right)^{-0.5} \end{bmatrix}. \quad (174)$$

Now since the quadratic term in the objective function equals $-0.5\mathbf{v}^T\mathbf{L}\mathbf{L}^T\mathbf{v}$, it is clear that a transformation on $\mathbf{v}$ that makes the Hessian equal to $-\mathbf{I}$, is $\mathbf{L}^T$ (since the quadratic term can be expressed as $-0.5(\mathbf{L}^T\mathbf{v})^T\mathbf{I}(\mathbf{L}^T\mathbf{v})$). Now let us suppose we have a transformation $\mathbf{W} = [\mathbf{A};\ \mathbf{b}]$. To transform its parameters into the space where the Hessian equals $-\mathbf{I}$, we need to transform the parameters with $\mathbf{L}^T$, i.e. the transpose of (173): for $1 \leq i \leq D$ and $1 \leq j < i$:

$$\begin{aligned}\tilde{w}_{ij} &= (1+d_j)^{0.5}w_{ij} + (1+d_j)^{-0.5}w_{ji} \quad (175)\\ \tilde{w}_{ji} &= \left(1+d_i-(1+d_j)^{-1}\right)^{0.5} w_{ji}, \quad (176)\end{aligned}$$

and for the diagonal of $\mathbf{A}$ we can scale by the square root of the appropriate term in (170): for $1 \leq i \leq D$,

$$\tilde{w}_{ii} = (2+d_i)^{0.5}w_{ii}. \quad (177)$$

The elements of $\mathbf{b}$, i.e. $w_{(D+1)i}$, are just copied. In order to transform the matrix back from this space to the original space, the reverse transformation is $\mathbf{L}^{-T}$, which is the transpose of (174): for $1 \leq i \leq D$ and $1 \leq j < i$:

$$\begin{aligned}w_{ij} &= (1+d_j)^{-0.5}\tilde{w}_{ij} \\ &\quad -\left(1+d_i-(1+d_j)^{-1}\right)^{-0.5}(1+d_j)^{-1}\tilde{w}_{ji} \quad (178)\\ w_{ji} &= \left(1+d_i-(1+d_j)^{-1}\right)^{-0.5}\tilde{w}_{ji} \quad (179)\end{aligned}$$

and for the diagonal,

$$w_{ii} = (2+d_i)^{-0.5}\tilde{w}_{ii}. \quad (180)$$

Suppose we have a gradient $\mathbf{P}$ with respect to the matrix parameters, where $\mathbf{P}$ has the same dimensions as $\mathbf{W}$ so that the objective function contains a term $\mathrm{tr}(\mathbf{W}\mathbf{P}^T)$. Transforming $\mathbf{P}$ to be in the transformed space $\mathbf{P} \rightarrow \tilde{\mathbf{P}}$ involves the inverse transpose of the "forward" transformation; this is clear as a general rule because if we have a vector $\mathbf{x}$ and a gradient $\mathbf{g}$ and we transform $\mathbf{x}$ with $\mathbf{M}$ and $\mathbf{g}$ with $\mathbf{M}^{-T}$, we have $(\mathbf{M}\mathbf{x})\cdot(\mathbf{M}^{-T}\mathbf{g}) = \mathbf{x}^T\mathbf{M}^T\mathbf{M}^{-T}\mathbf{g} = \mathbf{x}\cdot\mathbf{g}$. Therefore for $\mathbf{P} \rightarrow \tilde{\mathbf{P}}$ the transform $\mathbf{L}^{-1}$ applies: referring to (174), for $1 \leq i \leq D$ and $1 \leq j < i$,

$$\begin{aligned}\tilde{p}_{ij} &= (1+d_j)^{-0.5}p_{ij} \quad (181)\\ \tilde{p}_{ji} &= -\left(1+d_i-(1+d_j)^{-1}\right)^{-0.5}(1+d_j)^{-1}p_{ij} \\ &\quad +\left(1+d_i-(1+d_j)^{-1}\right)^{-0.5}p_{ji} \quad (182)\end{aligned}$$

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

$$\tilde{p}_{ii} = (2+d_i)^{-0.5}p_{ii}. \quad (183)$$

For the reverse transformation $\tilde{\mathbf{P}} \rightarrow \mathbf{P}$, the transform $\mathbf{L}$ applies: referring to Equation (173), for $1 \leq i \leq D$ and $1 \leq j < i$,

$$\begin{aligned}p_{ij} &= (1+d_j)^{0.5}\tilde{p}_{ij} \quad (184)\\ p_{ji} &= (1+d_j)^{-0.5}\tilde{p}_{ij} \\ &\quad +\left(1+d_i-(1+d_j)^{-1}\right)^{0.5}\tilde{p}_{ji} \quad (185)\end{aligned}$$

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

$$p_{ii} = (2+d_i)^{0.5}\tilde{p}_{ii}. \quad (186)$$

In Section 12.5 we will show how these transformations are used in the update rule.

12.4. Statistics accumulation

Here we describe the statistics accumulation for constrained MLLR estimation using this method.

We assume that we have done some kind of E-M to to get posteriors $\gamma_j(t)$ for each Gaussian j on each time t. We write the algorithm assuming we have a "flat" mixture of Gaussians, but this applies equally to a HMM; in that case the index j would range over the individual Gaussians in all the states of the HMM. If we are doing multiple passes over the data to estimate a transform and we are not on the first pass, the posteriors $\gamma_j(t)$ will be computed using the existing transform, but we accumulate statistics given the *original* features. We compute the following statistics:

$$\begin{aligned}\beta &= \sum_{t,j}\gamma_j(t) \quad (187)\\ \mathbf{K} &= \sum_{t,j}\gamma_j(t)\boldsymbol{\Sigma}_j^{-1}\boldsymbol{\mu}_j\mathbf{x}^+(t)^T \quad (188)\\ \mathbf{S}_j &= \sum_{t,j}\gamma_j(t)\mathbf{x}^+(t)\mathbf{x}^+(t)^T. \quad (189)\end{aligned}$$

and the auxiliary function is:

$$\begin{aligned}\mathcal{Q}(\mathbf{W}) &= \sum_{t,j}\gamma_j(t)\left(\log|\det\mathbf{A}|\right. \quad (190)\\ &\quad \left.-0.5(\mathbf{W}\mathbf{x}^+(t)-\boldsymbol{\mu}_j)^T\boldsymbol{\Sigma}_j^{-1}(\mathbf{W}\mathbf{x}^+(t)-\boldsymbol{\mu}_j)\right) \quad (191)\\ &= K+\beta\log|\det\mathbf{A}|+\mathrm{tr}(\mathbf{W}\mathbf{K}^T) \\ &\quad -0.5\sum_j\mathrm{tr}\left(\mathbf{W}^T\boldsymbol{\Sigma}_j^{-1}\mathbf{W}\mathbf{S}_j\right) \quad (192)\end{aligned}$$

12.5. Update

The update is an iterative update where on each iteration we compute the gradient of the objective function w.r.t. $\mathbf{W}$, use the pre-scaling described in the previous section to compute an update direction, and compute the optimal step size in that direction. On each iteration of update we refer to the current (pre-update) value of $\mathbf{W}$ as $\bar{\mathbf{W}}$. If we were using iteration indices we would write something like $\mathbf{W}^{(p-1)}$ instead of $\bar{\mathbf{W}}$. At the very start of the process $\bar{\mathbf{W}}$ equals $[\mathbf{I};\ \mathbf{0}]$; if we are not on the first iteration of the update or we are starting from an already estimated transform, this will not be the case. We derive the equations for a single iteration of update as follows. A linearization of (192) around $\mathbf{W} = \bar{\mathbf{W}}$ is:

$$\begin{aligned}\mathcal{Q}(\mathbf{W}) &\simeq K'+\beta\mathrm{tr}(\mathbf{A}\bar{\mathbf{A}}^{-1})+\mathrm{tr}(\mathbf{W}^T\mathbf{K}) \\ &\quad -\sum_j\mathrm{tr}\left(\mathbf{W}^T\left(\boldsymbol{\Sigma}_j^{-1}\bar{\mathbf{W}}\mathbf{S}_j\right)\right) \quad (193)\\ &\simeq K'+\mathrm{tr}\left(\mathbf{W}^T\mathbf{P}\right) \quad (194)\\ \mathbf{P} &= \beta\left[\mathbf{A}^{-T};\ \mathbf{0}\right]+\mathbf{K}-\mathbf{S}, \quad (195)\\ \mathbf{S} &= \sum_j\boldsymbol{\Sigma}_j^{-1}\bar{\mathbf{W}}\mathbf{S}_j \quad (196)\end{aligned}$$

where $\mathbf{M}^{+0}$ is $\mathbf{M}$ extended with an extra row of zeros. So $\mathbf{P}$ is the local gradient. In order to transform with $\mathbf{W}_{\mathrm{pre}}$ into the correctly normalized space, we can define $\mathbf{W}'$ as $\mathbf{W}$ in the pre-transformed space, so that we could take an original feature $\mathbf{x}$, and transform with $\mathbf{W}_{\mathrm{pre}}$, $\mathbf{W}'$ and then $\mathbf{W}_{\mathrm{inv}}$ (which is the inverse transformation to $\mathbf{W}_{\mathrm{pre}}$). It is convenient to turn all the transforms $\mathbf{W}$ into the form $\mathbf{W}^+$ which has an extra row whose last element is 1 and the rest are zero; this leaves a 1 on the end of the transformed vectors and makes $\mathbf{W}^+$ a square matrix. So we have:

$$\begin{aligned}\forall\mathbf{x},\ \mathbf{W}^+\mathbf{x}^+ &= \mathbf{W}_{\mathrm{inv}}^+\mathbf{W}'^+\mathbf{W}_{\mathrm{pre}}^+\mathbf{x}^+ \quad (197)\\ \mathbf{W}^+ &= \mathbf{W}_{\mathrm{inv}}^+\mathbf{W}'^+\mathbf{W}_{\mathrm{pre}}^+ \quad (198)\\ \mathbf{W}'^+ &= \mathbf{W}_{\mathrm{inv}}^{+\ -1}\mathbf{W}^+\mathbf{W}_{\mathrm{pre}}^{+\ -1} \quad (199)\\ \mathbf{W}' &= \mathbf{W}_{\mathrm{pre}}\mathbf{W}^+\mathbf{W}_{\mathrm{inv}}^+ \quad (200)\end{aligned}$$

We can use this to transform $\mathbf{P}$ to $\mathbf{P}'$ to apply in the transformed space. Again it is useful to represent everything as square matrices:

$$\begin{aligned}\forall\mathbf{W},\ \mathrm{tr}(\mathbf{W}^T\mathbf{P}) &= \mathrm{tr}(\mathbf{W}'^T\mathbf{P}') \quad (201)\\ \forall\mathbf{W},\ \mathrm{tr}(\mathbf{W}^T\mathbf{P}) &= \mathrm{tr}(\mathbf{W}_{\mathrm{inv}}^{+\ T}\mathbf{W}^{+T}\mathbf{W}_{\mathrm{pre}}^T\mathbf{P}') \quad (202)\\ \forall\mathbf{W},\ \mathrm{tr}(\mathbf{W}^{+T}\mathbf{P}^{+0}) &= \mathrm{tr}(\mathbf{W}^{+T}\mathbf{W}_{\mathrm{pre}}^T\mathbf{P}'\mathbf{W}_{\mathrm{inv}}^{+\ T}) \quad (203)\\ \mathbf{P}^{+0} &= \mathbf{W}_{\mathrm{pre}}^T\mathbf{P}'\mathbf{W}_{\mathrm{inv}}^{+\ T} \quad (204)\end{aligned}$$

At this point we use the fact that $\mathbf{W}_{\mathrm{inv}}^{+\ T-}\mathbf{W}_{\mathrm{pre}}^T = \mathbf{I}$. This is true because $\mathbf{A}^-\mathbf{B}^{-C} = (\mathbf{A}\mathbf{B})^{--}$, with $\cdot^-$, $\cdot^{-C}$ and $\cdot^{--}$ meaning removing respectively the last row, the last column, and both; and we

use this together with the fact that $\mathbf{W}_{\text{inv}}^{+\ T}\mathbf{W}_{\text{pre}}^{+\ T} = \mathbf{I}$. We can then arrive at:

$$\begin{aligned}\mathbf{P}' &= \mathbf{W}_{\text{inv}}^{+\ T^-}\mathbf{P}^{+0}\mathbf{W}_{\text{pre}}^{+\ T}. && (205)\\ &= \left[\mathbf{A}_{\text{pre}}^{-T};\ 0\right]\mathbf{P}^{+0}\mathbf{W}_{\text{pre}}^{+\ T} && (206)\\ &= \mathbf{A}_{\text{inv}}^{T}\mathbf{P}\mathbf{W}_{\text{pre}}^{+\ T},, && (207)\end{aligned}$$

where $\mathbf{A}_{\text{inv}} = \mathbf{A}_{\text{pre}}^{-1}$ is the first d columns of $\mathbf{W}_{\text{inv}}$. Note that (207) is similar to the inversed and transposed equivalent of (200), which is what we expect for a quantity and its derivative. After computing $\mathbf{P}'$ using (207), which means we have applied the pre-transform to the gradient, we apply the transformation of Equations (181) to (183), which gives us a quantity we can call $\tilde{\mathbf{P}}$. In this space the proposed change $\mathbf{\Delta}$ in $\mathbf{W}$ will be:

$$\tilde{\mathbf{\Delta}} = \frac{1}{\beta}\tilde{\mathbf{P}}. \quad (208)$$

We then transform $\tilde{\mathbf{\Delta}}$ to $\mathbf{\Delta}'$ using Equations (178) to (180); note that $\mathbf{\Delta}$ takes the place of $\mathbf{W}$ in those equations. Then we transform $\mathbf{\Delta}'$ to $\mathbf{\Delta}$; referring to Equation (198):

$$\mathbf{\Delta} = \mathbf{W}_{\text{inv}}\mathbf{\Delta}'^{+0}\mathbf{W}_{\text{pre}}^{+}. \quad (209)$$

At this point a useful check to do is to make sure that:

$$\begin{aligned}\text{tr}\left(\mathbf{\Delta}\mathbf{P}^T\right) &= \text{tr}\left(\mathbf{\Delta}'\mathbf{P}'^T\right) && (210)\\ &= \text{tr}\left(\tilde{\mathbf{\Delta}}\tilde{\mathbf{P}}^T\right). && (211)\end{aligned}$$

This is a check that the co-ordinate transformations have been consistently done. At this point we have a suggested change $\mathbf{\Delta}$ in $\mathbf{W}$ in the original co-ordinates, which in most cases we should be able to apply without problems. But we are still not guaranteed to increase the objective function. At this point we decide to make a step $k\mathbf{\Delta}$, where k will be close to 1 if our approximations are accurate, and we will choose k to maximize the auxiliary function.

Referring to the auxiliary function in Equation (192), and using $\mathbf{W} = \tilde{\mathbf{W}} + k\mathbf{\Delta}$, we can express the auxiliary function as a function of k (ignoring constant terms) as:

$$\begin{aligned}\mathcal{Q}(k) &= \beta\log|\det\mathbf{A}| + k\,\text{tr}\left(\mathbf{\Delta}\mathbf{K}^T\right) && (212)\\ &\quad -k\,\text{tr}\left(\mathbf{\Delta}\mathbf{S}^T\right) - 0.5k^2\sum_j \text{tr}\left(\mathbf{\Delta}^T\mathbf{\Sigma}_j^{-1}\mathbf{\Delta}\mathbf{S}_j\right), && (213)\end{aligned}$$

with $\mathbf{S}$ as defined in Equation (196). We will iteratively optimize the scalar k using Newton's method, starting at $k = 0$. First we simplify the auxiliary function in k as below, using $\mathbf{\Delta}^{-C}$ to mean $\mathbf{\Delta}$ with its last column removed:

$$\begin{aligned}\mathcal{Q}(k) &= \beta\log\det(\mathbf{A} + k\mathbf{\Delta}^{-C})\\ &\quad + k\,m - 0.5k^2 n, && (214)\\ m &= \text{tr}\left(\mathbf{\Delta}\mathbf{K}^T\right) - \text{tr}\left(\mathbf{\Delta}\mathbf{S}^T\right) && (215)\\ n &= \sum_j \text{tr}\left(\mathbf{\Delta}^T\mathbf{\Sigma}_j^{-1}\mathbf{\Delta}\mathbf{S}_j\right). && (216)\end{aligned}$$

On each iteration of optimizing, we need the first and second derivatives of the auxiliary function with respect to k. We can compute:

$$\begin{aligned}\frac{\mathbf{d}\mathcal{Q}(k)}{\mathbf{d}k} &= \beta\text{tr}\left((\mathbf{A} + k\mathbf{\Delta}^{-C})^{-1}\mathbf{\Delta}^{-C}\right)\\ &\quad + m - kn && (217)\\ \frac{\mathbf{d}^2\mathcal{Q}(k)}{\mathbf{d}k^2} &= -\beta\text{tr}\left((\mathbf{A} + k\mathbf{\Delta}^{-C})^{-1}\mathbf{\Delta}^{-C}(\mathbf{A} + k\mathbf{\Delta}^{-C})^{-1}\mathbf{\Delta}^{-C}\right)\\ &\quad - n, && (218)\end{aligned}$$

with derivations of the parts of Equations (217) and (218) that involve matrices put in Appendix E. The update is:

$$\hat{k} = k + \frac{\delta\mathcal{Q}(k)/\delta k}{-\delta^2\mathcal{Q}(k)/\delta k^2}. \quad (219)$$

On each of these iterations we should compute the value of Equation (214) to check that it is not decreasing, and in that case keep halving the change in k until it is not decreasing. This situation should rarely happen. Updating k for five or ten iterations should be sufficient; the time taken to do this does not dominate the computation.

The final value of k should be reasonably close to 1. It may be helpful to print out the optimal values of k on each iteration as a sanity check on the algorithm. Each step (i.e. each time we calculate a $\mathbf{\Delta}$ and estimate the optimal k), the value of Equation (214) given the optimal k is the objective function change. We can see this because $k = 0$ corresponds to no change in $\mathbf{W}$, and in that case Equation (214) is zero. After we estimate k, the update is:

$$\mathbf{W} \leftarrow \mathbf{W} + k\mathbf{\Delta}. \quad (220)$$

The overall process has three levels of iteration. The outer level is where we accumulate statistics using Equations (187) to (189), where typically one or two iterations should suffice. The intermediate level is where each time we compute a change $\mathbf{\Delta}$, iteratively optimize k and update the matrix $\mathbf{W}$, using roughly Equations (195) to (220); we expect to do perhaps 5 to 10 of these iterations. The inner level is the number of iterations needed to estimate k, where also perhaps 5 to 10 iterations can be used but this choice is not very critical.

13. SUBSPACE VERSION OF CONSTRAINED MLLR

Let us suppose we have a set of "basis" constrained MLLR matrices $\mathbf{W}_b$ for $1 \leq b \leq B$, and we force our estimated matrix $\mathbf{W}$ to have the form:

$$\mathbf{W}^{(s)} = \mathbf{W}_0 + \sum_{b=1}^{B} d_b^{(s)}\mathbf{W}_b, \quad (221)$$

where $\mathbf{W}_0 = [\mathbf{I}; \mathbf{0}]$, and we include this to ensure the "default" transform is in our subspace. We are borrowing some notation from [8]. It is actually more convenient to express this relationship in the fully transformed space:

$$\tilde{\mathbf{W}}^{(s)} = \tilde{\mathbf{W}}_0 + \sum_{b=1}^{B} d_b^{(s)}\tilde{\mathbf{W}}_b. \quad (222)$$

The transform $\tilde{\mathbf{W}}_0$ does not have the simple form of $\mathbf{W}_0$ because when we change co-ordinates we scale the diagonal of $\mathbf{A}$, but we never have to refer to $\tilde{\mathbf{W}}_0$ in our calculation so we don't write down the expression for it. It is useful to represent the set of $\tilde{\mathbf{W}}_b$ as an orthonormal basis, so that:

$$\begin{aligned}\text{tr}\left(\tilde{\mathbf{W}}_b\tilde{\mathbf{W}}_b^T\right) &= 1 && (223)\\ \text{tr}\left(\tilde{\mathbf{W}}_b\tilde{\mathbf{W}}_c^T\right) &= 0,\ c \neq b. && (224)\end{aligned}$$

Note that $\text{tr}\left(\mathbf{A}\mathbf{B}^T\right)$ is the the same as the dot product of the concatenated rows (or columns) of the matrices $\mathbf{A}$ and $\mathbf{B}$, and Equations (223) and (224) make the most sense while thinking about the

matrices as vectors. In the baseline approach, the update in the transformed space was very simple:

$$\tilde{\mathbf{\Delta}} = \frac{1}{\beta}\tilde{\mathbf{P}}, \quad (225)$$

to repeat Equation (208). With the subspace approach, we would do instead:

$$\tilde{\mathbf{\Delta}} = \frac{1}{\beta}\sum_{b=1}^{B}\tilde{\mathbf{W}}_b \mathrm{tr}\,(\tilde{\mathbf{W}}_b\tilde{\mathbf{P}}^T). \quad (226)$$

Note that in this method of calculation the quantities $d_b^{(s)}$ are "implicit" and are never referred to in the calculation, but the updated $\mathbf{W}$ will still be constrained by the subspace. This makes it possible to do more code sharing with the non-subspace-constrained version of the Constrained MLLR computation and simplifies a lot of procedures, but at the cost of memory and possibly disk space. The use of the subspace will not slow down the update significantly if B is not much larger than D and if we compute $\mathrm{tr}\,(\tilde{\mathbf{W}}_b\tilde{\mathbf{P}}^T)$ in a reasonable way (without actually computing the matrix product).

13.1. Training the basis

Training the basis is quite straightforward. Assuming our Hessian is correct and the update is reasonably small, we can compute our auxiliary function improvement in the transformed space as $0.5\mathrm{tr}\,(\tilde{\mathbf{\Delta}}\tilde{\mathbf{P}}^T)$. This is is the same as $0.5\mathrm{tr}\,(\frac{1}{\sqrt{\beta}}\tilde{\mathbf{P}}\frac{1}{\sqrt{\beta}}\tilde{\mathbf{P}}^T)$, which is half the sum of squared elements of $\frac{1}{\sqrt{\beta}}\tilde{\mathbf{P}}$. If we are using a subspace, our auxiliary function improvement is the trace of the scatter of this quantity projected into the subspace. So the basis computation consists of computing $\frac{1}{\sqrt{\beta^{(s)}}}\tilde{\mathbf{P}}^{(s)}$ for all speakers s, turning each matrix into a vector by concatenating the rows and then computing the top eigenvectors of the scatter of this quantity. Note that in order to do this, it is not necessary to actually compute the scatter. It can be more efficient to do the eigenvalue computation on the vectors themselves, as described Appendix A. The associated eigenvalues are useful for diagnostics; we expect the eigenvalues to drop off quite rapidly. The quantity $\frac{1}{\sqrt{\beta}}\tilde{\mathbf{P}}^{(s)}$ is computed once for all speakers without actually doing any adaptation: we compute it for each training speaker s as if we are about to start optimizing $\mathbf{W}^{(s)}$ for that speaker.

13.2. Interaction with class-based constrained MLLR

In class-based constrained MLLR, we apply a different transform for different sets ("classes") of the Gaussians in a system. In this case it would most likely be beneficial to compute all the generic parameters, such as $\mathbf{W}_{\mathrm{inv}}$, $\mathbf{D}$ and the basis matrices $\tilde{\mathbf{W}}_b$, separately for each class. It is common to do regression tree based MLLR, in which the classes are arranged into a binary tree (the original classes are at the leaves), and the transform is estimated at nodes of the tree, not necessarily leaf nodes, where there is enough data. This corresponds to hierarchically merging similar classes. These generic parameters should probably be estimated at all nodes of the tree in that case. In the recipe we have in mind for the subspace-based system, there will probably be about 10 regression classes (based on the "sub-models" $1 \leq k \leq K$). We will most likely just enable computation for these regression classes, and one "global" class corresponding to the merge of all of them. Because there are very few parameters to estimate per speaker, there is not much point in going to the trouble of coding a regression tree since in most cases we will have enough data to estimate transforms at its leaves.

14. SPEAKER TRANSFORMS (CONSTRAINED MLLR)

In this section we summarize the various forms of update associated with constrained MLLR estimation. This section simply summarizes how to use the techniques described in Sections 12 and 13 in the context of a factor analyzed GMM. The reader who is simply trying to understand the fundamental ideas may find it best to skip this section.

14.1. Phases of transform computation

There are three different phases involved in the computation for the subspace-based speaker transforms. The first phase is the computation of the pre-transforms and associated parameters, both global and sub-model-specific versions. These are quantities that appear in our update equations for Constrained MLLR estimation. This phase should probably take place after we have already begun training the subspace based model, but before we have begun training it speaker adaptively (using the speaker vectors $\mathbf{v}_k^{(s)}$ and projections $\mathbf{N}_{ki}$). The reason is that to estimate these parameters we need a model that corresponds as much as possible to the model we are going to use in testing, but the incorporation of the other form of speaker adaptation in this phase would be very complicated, and would probably not help. The second phase is the estimation of the subspace– i.e. the basis matrices $\tilde{\mathbf{W}}_b$, both globally for the sub-models. This requires us to accumulate statistics for each speaker as if we are going to estimate transforms, but instead just compute a gradient term that we store for each speaker; the subspace is computed by finding the top eigenvectors of the scatter of these quantities. We will compute a subspace separately for each sub-model k and also a global one for back-off when there is very little data. The third phase is where we already have the pre-transforms and subspaces, and are ready to compute speaker transforms. At this point we start estimating transforms for training speakers, and the other parts of accumulation take place on top of the transformed features.

Note that these three different phases would be interleaved with other phases of model training; they would each be done on particular iterations of the model training. For example, supposing we have 20 iterations of model training overall, we might decide that phase 1 (pre-transforms) takes place on iteration 5, phase two (basis estimation) takes place on iteration 6, and transform estimation (phase three) takes place on iteration 7 and every 5'th iteration thereafter. Typically other forms of estimation would take place on those iterations also.

The order of estimating the constrained MLLR transforms and the factor-based adaptation ($\mathbf{v}_k^{(s)}$ and $\mathbf{N}_{ki}$) needs to be decided. They should probably be estimated in the same order in training and testing. There is also the question of whether to reset the adaptation parameters prior to re-estimating them each time. These decisions do not affect the equations, since we write each form of update assuming the other is already in place, and if not the other will just take some default value that implies no adaptation is taking place.

14.2. Pre-transform computation

The pre-transform computation is something that can be done statically from the models only, without reference to the data. We will generally choose a particular iteration of update, typically near the beginning, on which to do this. The output of this phase is as follows. First we have a global pre-transform pair $\mathbf{W}_{\mathrm{pre}}$ and $\mathbf{W}_{\mathrm{inv}}$ with its associated diagonal projected mean-scatter $\mathbf{D}$ (we just store the diagonal elements of this). Then we have the same for each sub-model

k; we can call these $\mathbf{W}_{\text{pre}}^{(k)}$, $\mathbf{W}_{\text{inv}}^{(k)}$, and $\mathbf{D}^{(k)}$.

For this phase we need to have a prior for each acoustic state j, and we can either just assume a flat prior $\gamma_j = 1/J$ or estimate it from counts. Then we can compute:

$$\gamma_k = \sum_{j,m} \gamma_j c_{jkm} \tag{227}$$

$$\mathbf{\Sigma}_W^{(k)} = \frac{1}{\gamma_k} \sum_{j,m,i} \gamma_j c_{jkm} w_{jkmi} \mathbf{\Sigma}_{ki} \tag{228}$$

$$\boldsymbol{\mu}^{(k)} = \frac{1}{\gamma_k} \sum_{j,m,i} \gamma_j c_{jkm} w_{jkmi} \boldsymbol{\mu}_{jkmi} \tag{229}$$

$$\mathbf{\Sigma}_B^{(k)} = \frac{1}{\gamma_k} \left(\sum_{j,m,i} \gamma_j c_{jkm} w_{jkmi} \boldsymbol{\mu}_{jkmi} \boldsymbol{\mu}_{jkmi}^T \right) - \boldsymbol{\mu}^{(k)} \, \boldsymbol{\mu}^{(k)\,T}. \tag{230}$$

It may be helpful to prune away very small counts (i.e. if $\gamma_j c_{jkm} w_{jkmi}$ is very small) to avoid computing the mean $\boldsymbol{\mu}_{jkmi}$ or its outer product, both of which require some computation. The pre-transform computation starting from these statistics is described in Section 12.2.

To compute the global version of the required quantities, we can average the sub-model-dependent statistics with:

$$\mathbf{\Sigma}_W = \sum_k \gamma_k \mathbf{\Sigma}_W^{(k)} \tag{231}$$

$$\boldsymbol{\mu} = \sum_k \gamma_k \boldsymbol{\mu}^{(k)} \tag{232}$$

$$\mathbf{\Sigma}_B = \sum_k \gamma_k \left(\mathbf{\Sigma}_B^{(k)} + \boldsymbol{\mu}^{(k)} \boldsymbol{\mu}^{(k)\,T} \right) - \boldsymbol{\mu} \boldsymbol{\mu}^T. \tag{233}$$

Again, the pre-transform computation is as described in Section 12.2.

14.3. Basis computation

Next we need to compute the basis matrices; these include the global basis matrices $\tilde{\mathbf{W}}_b$ for $1 \leq b \leq B$, and the sub-model basis matrices $\tilde{\mathbf{W}}_b^{(k)}$. In order to do this, for each speaker s we need to compute statistics $\beta_k^{(s)}$, $\mathbf{K}_k^{(s)}$ and $\mathbf{S}_{ki}^{(s)}$. This is as described in Equations (64) to (66). We can just sum these statistics up to get $\beta^{(s)}$ and $\mathbf{K}_k^{(s)}$ which apply to the global (not sub-model-specific) version of the computation. The global version of the computation uses the statistics $\mathbf{S}_{ki}^{(s)}$ for all values of k.

Using these statistics for speaker s we need to compute the global matrix of gradients $\tilde{\mathbf{P}}^{(s)}$ for each speaker s and the sub-model-specific ones $\tilde{\mathbf{P}}_k^{(s)}$. Using the global version to illustrate the process, we compute $\mathbf{P}^{(s)}$ with Equations (195) and (196), pre-transform it to $\mathbf{P}'^{(s)}$ using Equation (207), and to $\tilde{\mathbf{P}}^{(s)}$ with Equations (181) to (183). We then scale by $\frac{1}{\sqrt{\beta}}$ and store the result.

The outcome of the above process is that we have a set of matrices for each speaker $1 \leq s \leq S$, namely the global version $\frac{1}{\sqrt{\beta^{(s)}}} \tilde{\mathbf{P}}^{(s)}$, and also the sub-model specific versions $\frac{1}{\sqrt{\beta^{(s)}}} \tilde{\mathbf{P}}_k^{(s)}$. The rest of the basis computation is simple: we turn each matrix into a vector, compute the scatter of these vectors, and then globally and for each k we compute the top eigenvectors and turn the result back into sets of basis matrices. To make this explicit, supposing $\text{vec}(\mathbf{A})$ means concatenating together the rows of $\mathbf{A}$, for a particular k we would compute the scatter matrix $\mathbf{M} = \sum_{s \in \mathcal{S}} \text{vec}(\frac{1}{\sqrt{\beta^{(s)}}} \tilde{\mathbf{P}}^{(s)}) \text{vec}(\frac{1}{\sqrt{\beta^{(s)}}} \tilde{\mathbf{P}}^{(s)})^T$, do an eigenvalue decomposition $\mathbf{M} = \mathbf{V}\mathbf{D}\mathbf{V}^T$, and assuming the columns of $\mathbf{V}$ are sorted from highest eigenvalue to lowest we could take the first B columns $\mathbf{v}_b$, $1 \leq b \leq B$ of $\mathbf{V}$ as our basis, turning each $\mathbf{v}_b$ into a matrix $\tilde{\mathbf{W}}_b$ by making each block of $D+1$ elements a row of the matrix. The associated eigenvalues (the diagonal of $\mathbf{D}$) should also be useful for diagnostic purposes as they tell us how much of the speaker variation is present in each dimension.

14.4. Speaker transform computation

The computation of the speaker transforms needs the statistics $\beta_k^{(s)}$, $\mathbf{K}_k^{(s)}$ and $\mathbf{S}_{ki}^{(s)}$ which are accumulated as described in Equations (64) to (66). The update is as described in Section 12.5 with the subspace modification described in Section 13. If any sub-model k has less than some specified minimum count (e.g. twice the subspace dimension), we can back off to a global version of the transform which may be estimated by summing up the statistics $\beta_k^{(s)}$ and $\mathbf{K}_k^{(s)}$ to get a globally summed version of the statistics, and doing the same computation.

15. REFERENCES

[1] Pelecanos J. Povey D., Chu S. M., "Approaches to Speech Recognition based on Speaker Recognition Techniques," Book chapter in forthcoming book on GALE project, 2009.

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

[3] Dehak N. Kenny P., Ouellet P. and Gupta V., "A study of Interspeaker Variability in Speaker Verification," *IEEE Trans. on Audio, Speech and Language Processing*, vol. 16, no. 5, pp. 980–987, 2008.

[4] Kuhn R., Junqua J.-C., P. Nguyen, and N. Niedzielski, "Rapid Speaker Adaptation in Eigenvoice Space," *IEEE Transactions on Speech and Audio Processing*, vol. 8, no. 6, Nov. 2000.

[5] M. J. F. Gales, "Multiple-cluster adaptive training schemes," in *ICASSP*, 2001.

[6] Chu S. M. Povey D. and B. Varadarajan, "Universal Background Model Based Speech Recognition," in *ICASSP*, 2008.

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

[8] Visweswariah K., Goel V., and Gopinath R., "Structuring Linear Transforms for Adaptation Using Training Time Information," in *ICASSP*, 2002.

[9] Sim K C. and Gales Mark J. F., "Adaptation of Precision Matrix Models on Large Vocabulary Continuous Speech Recognition," in *ICASSP*, 2005.

[10] Saon G. Povey D., "Feature and model space speaker adaptation with full covariance Gaussians," in *Interspeech/ICSLP*, 2006.

[11] M.J.F Gales, "Maximum likelihood linear transformations for hmm-based speech recognition," *Computer Speech and Language*, vol. 12, 1998.

[12] Golub and van Loan, *Matrix computations*, Johns Hopkins University Press, 3 edition, 1983.

[13] Cullum and Willoughby, *Lanczos Algorithms for Large Symmetric Eigenvalue Computations*.

[14] Paliwal K. K. Sharma A., "Fast principal component analysis using fixed-point algorithm," *Pattern Recognition Letters*, vol. 28, pp. 1151–1155, 2007.

APPENDICES

A. FAST TOP-N-EIGENVALUE COMPUTATION

Many situations arise where we need to compute the top N eigenvalues and eigenvectors of a symmetric matrix $\mathbf{M} = \sum_{k=1}^{K} \mathbf{v}_k \mathbf{v}_k^T$, with $\mathbf{v}_k \in \Re^D$. We will assume that $N \ll D$ and $N \ll K$ (i.e. the number of eigenvalues we need is quite small compared with the other quantities), but the number of vectors K may be greater or less than D.

This problem can be solved via SVD. The most obvious method would be to compute $\mathbf{M}$ and do the singular value decomposition $\mathbf{M} = \mathbf{U}\mathbf{D}\mathbf{V}^T$, sort the columns of $\mathbf{U}$ and the corresponding diagonal elements of $\mathbf{D}$ from largest to smallest (SVD implementations generally do not do this sorting exactly so the user must do it), and return the first N columns of $\mathbf{U}$ as the top eigenvectors and the first N diagonal elements of $\mathbf{D}$ as the corresponding eigenvalues.

If $K < D$, we can solve this more efficiently: let $\mathbf{A}$ be the D by K (tall) matrix whose k'th column is $\mathbf{v}_k$, so that $\mathbf{M} = \mathbf{A}\mathbf{A}^T$. We can work out the SVD of $\mathbf{M}$ via SVD on a smaller K by K matrix, as follows. First, compute $\mathbf{A}^T\mathbf{A}$ and do the singular value decomposition $\mathbf{A}^T\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T$. Because $\mathbf{A}^T\mathbf{A}$ is bound to be positive semi-definite we can show that $\mathbf{A}^T\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{U}^T$ (i.e. we can discard $\mathbf{V}$). At this point we can write down $\mathbf{M} = \mathbf{A}\mathbf{A}^T = (\mathbf{A}\mathbf{U}\mathbf{D}^{-0.5})\mathbf{D}(\mathbf{D}^{-0.5}\mathbf{U}^T\mathbf{A}^T) = \mathbf{W}\mathbf{D}\mathbf{W}^T$, where the columns of $\mathbf{W} = \mathbf{A}\mathbf{U}\mathbf{D}^{-0.5}$ are orthonormal because $\mathbf{W}^T\mathbf{W} = \mathbf{D}^{-0.5}\mathbf{U}^T\mathbf{A}^T\mathbf{A}\mathbf{U}\mathbf{D}^{-0.5} = \mathbf{I}$. This is the same as singular value decomposition $\mathbf{M}$ with $\mathbf{W}$ containing only the first K columns, and and we could easily construct the full SVD from by extending $\mathbf{W}$ [12, Theorem 2.5.1]. So as before we sort the diagonal elements of $\mathbf{D}$ and the corresponding columns of $\mathbf{W}$ from largest to smallest eigenvalue, and return the top N. This argument does not cover the case when elements of $\mathbf{D}$ are zero, but extending it to cover that case is not difficult.

A simpler but less efficient way of doing it in the case where $K < D$, is to do the singular value decomposition $\mathbf{A} = \mathbf{U}\mathbf{L}\mathbf{V}^T$ (assume this is the "skinny" SVD so $\mathbf{U} \in \Re^{D\times K}$, $\mathbf{L} \in \Re^{K\times K}$ and $\mathbf{V} \in \Re^{K\times K}$), so $\mathbf{M} = \mathbf{U}\mathbf{L}^2\mathbf{U}^T$. The diagonal elements of $\mathbf{L}$ will always be positive (this is how SVD is defined), so we only have to sort the diagonal elements of $\mathbf{L}$ and the corresponding rows of $\mathbf{U}$ as before and the answer is the first N columns of $\mathbf{U}$ and the *squares* of the first N diagonal elements of $\mathbf{L}$.

The methods described above should be reasonably fast, e.g. SVD on $\mathbf{M}$ where $D = 40 \cdot 41 = 1640$ as we would encounter in the CMLLR computation with 40 dimensional data should take about $12D^3 = 53\times10^9$ floating point operations [12, Section 5.4.4], which at one Gigaflop would take about one minute. So it is quite possible to do this in the standard way. However, it is possible to do it much faster as described below.

A.1. Fast top N eigenvalue computation

This section describes an algorithm for quickly computing the N eigenvalues with the largest absolute value and their corresponding eigenvectors, for a symmetric $D \times D$ matrix $\mathbf{M}$, with $D \gg N$. The technique used here is a form of the Lanczos method [12, 13]. It is also related to the technique described in [14] (and is solving the same problem), but it is faster.

Let us formulate the problem as saying we need an orthonormal set of vectors $\mathbf{a}_n$, $1 \le n \le N$, such that if the columns of $\mathbf{A}$ are $\mathbf{a}_n$ and $\mathbf{B} = \mathbf{A}^T\mathbf{M}\mathbf{A}$, $\mathbf{B}$ is diagonal and the sum of absolute values of the diagonal elements of $\mathbf{B}$ is maximized. This is basically the same problem we have when doing PCA and related methods. An exact solution to this could be obtained by making $\mathbf{a}_i$ the N eigenvectors with the largest magnitude eigenvalues. Typically in the kinds of problems that this arises in, we will only be dealing with positive eigenvalues, but in order for this approach to work we have to assume we need the largest *absolute* eigenvalues; this is of no practical consequence in most cases.

In this algorithm, we construct a sequence of vectors $\mathbf{v}_i$ ($1 \le i \le I$) which form an orthonormal basis ($\mathbf{v}_i \cdot \mathbf{v}_j = \delta_{ij}$), and a sequence $\mathbf{u}_i$ ($1 \le i \le I$) such that $\mathbf{u}_i = \mathbf{M}\mathbf{v}_i$. If we have $\mathbf{U}$ and $\mathbf{V}$ such that the rows of $\mathbf{U}$ are $\mathbf{u}_i$ and the rows of $\mathbf{V}$ are $\mathbf{v}_i$, we have $\mathbf{U}^T = \mathbf{M}\mathbf{V}^T$. If we then compute the $I \times I$ matrix $\mathbf{W} = \mathbf{V}\mathbf{U}^T$, then $\mathbf{W} = \mathbf{V}\mathbf{M}\mathbf{V}^T$. $\mathbf{W}$ is just $\mathbf{M}$ restricted to a subspace defined by the vectors $\mathbf{v}_i$. Within this restricted subspace, we can solve our problem as defined above by taking the top D eigenvectors of $\mathbf{W}$ and representing them appropriately in the original space. The problem then is to get a restricted subspace that contains as much as possible of the variance within $\mathbf{M}$. The basic idea is to start with a random vector $\mathbf{r}$ and keep multiplying by $\mathbf{M}$ and orthogonalizing. At the end the rows of $\mathbf{V}$ will be some linear combination of $\mathbf{r}, \mathbf{M}\mathbf{r} \ldots \mathbf{M}^{I-1}\mathbf{r}$. This process will tend to lead to a subspace dominated by the eigenvectors of $\mathbf{M}$ with the largest eigenvalues. The theory behind this is quite complicated; see the chapter on Lanczos methods in [12] for a discussion and references.

The way it is done in the algorithm we write below, we never have to store $\mathbf{U}$ because we construct $\mathbf{W}$ as we go along and $\mathbf{W}$ contains all the information needed to construct $\mathbf{U}$ from $\mathbf{V}$. $\mathbf{W}$ has a tri-diagonal structure and this could in principle be used to reduce the number of vector-vector multiplies we do: we dot $\mathbf{u}_i$ with all $\mathbf{v}_j$, $1 \le j \le i$, but it is not necessary for $j < i - 1$. However, avoiding those dot products leads to numerical instability and they do not dominate the computation anyway. We also waste space and time by making no use of the tri-diagonal nature of $\mathbf{W}$ in its storage or in the singular value or eigenvalue decomposition of $\mathbf{W}$ (see [12] for methods to do this), but these things do not dominate the overall memory or time of the computation.

The algorithm is as follows. Let the number of iterations I equal $\min(N + X, D)$, with for instance $X = 40$; X is the number of extra iterations. We only get a saving by using this approach if $I \ll D$. We set $\mathbf{v}_1$ to be a random unit vector, and $\mathbf{W}$ to be an I by I matrix whose elements are initially all zero. $\mathbf{W}$ represents a sequence of vectors $\mathbf{u}_i$ in the orthonormal basis represented by $\mathbf{v}_i$; each $\mathbf{u}_i$ equals $\mathbf{M}\mathbf{v}_i$. On each iteration we set $\mathbf{u}_i$ to $\mathbf{M}\mathbf{v}_i$ and set $\mathbf{v}_{i+1}$ to be the direction in $\mathbf{u}_i$ that is orthogonal to all previous $\mathbf{v}_i$. The algorithm is:

Set $\mathbf{v}_1$ to be a random, unit-length vector.
For each iteration $1 \leq i \leq I$:
$\mathbf{v}_{i+1} \leftarrow M\mathbf{v}_i$
for $1 \leq j \leq i$,
$\mathbf{W}_{ij} \leftarrow \mathbf{v}_{i+1} \cdot \mathbf{v}_j$
$\mathbf{v}_{i+1} = \mathbf{v}_{i+1} - \mathbf{v}_j \mathbf{W}_{ij}$
if $j < i - 1$,
Check that $\mathbf{W}_{ij}$ is small; set it to exactly zero
(in this case we only did it for numerical stability)
if $i < I$,
$\mathbf{W}_{i,i+1} \leftarrow |\mathbf{v}_{i+1}|$
$\mathbf{v}_{i+1} \leftarrow \frac{1}{|\mathbf{v}_{i+1}|}\mathbf{v}_{i+1}$
(where $|\mathbf{x}| = \sqrt{\sum_i x_i}$)
Make sure the tri-diagonal matrix $\mathbf{W}$ is exactly symmetrical
(symmetrize in case of small errors)
Use a standard method to do the eigenvalue decomposition of $\mathbf{W}$
as $\mathbf{W} = \mathbf{PDP}^T$
Sort $\mathbf{P}$ and $\mathbf{D}$ in terms of the *absolute* eigenvalue, by
rearranging the columns of $\mathbf{P}$ and the diagonal elements of $\mathbf{D}$.
Let $\mathbf{P}_{N,I}^T$ equal $\mathbf{P}^T$ truncated to dimension $N \times I$.
Let $\mathbf{V}$ be a I by D matrix whose rows are $\mathbf{v}_i$, $1 \leq i \leq I$.
The top N eigenvectors we return are the rows of $\mathbf{P}_{N,I}^T\mathbf{V}$
The top N eigenvalues are the first N diagonal elements of $\mathbf{D}$.

Given the way we have formulated the problem above, a sensible testing approach is as follows. If the routine returns a vector $\mathbf{A}$ whose rows are the (approximate) eigenvectors, we can test it as follows. $\mathbf{A}$ should be orthonormal ($\mathbf{A}^T\mathbf{A} = \mathbf{I}$), and we should evaluate $\mathbf{B} = \mathbf{AMA}^T$. $\mathbf{B}$ should be diagonal, and the sum of the absolute values of its diagonal elements should be very close to the sum of the N largest eigenvalues of $\mathbf{M}$ (it cannot be more than that).

We can check that we have set the number of iterations I large enough by evaluating the sum of absolute diagonal elements of $\mathbf{B}$ (or equivalently, the largest D elements of $\mathbf{D}$ inside the algorithm) and testing whether it seems to be converging as we increase I. The number of required iterations will depend on how closely spaced the matrix's eigenvalues are.

An extension of this that can be used when the large matrix $\mathbf{M}$ consists of a sum of outer products $\mathbf{M} = \sum_{k=1}^K \mathbf{m}_i\mathbf{m}_i^T$ is to do the multiplication $\mathbf{u}_i \leftarrow \mathbf{M}\mathbf{v}_i$ as $\mathbf{u}_i \leftarrow \sum_{k=1}^K (\mathbf{v}_i \cdot \mathbf{m}_i)\mathbf{m}_i$. If we take into account the time used to construct the matrix $\mathbf{M}$, this modification will be faster whenever (approximately) $2IKD < (K+I)D^2$, where I is the number of iterations. If $I > K$ then we can set $I = K$ because the extra iterations will not help. Therefore it is sufficient to show that $2IKD < 2KD^2$. This reduces to $I < D$, so given that we only claim that this technique is useful when $I \ll D$, this modification will always help where the overall technique is applicable.

B. SINGULAR VALUE DECOMPOSITION IN THE SYMMETRIC POSITIVE DEFINITE CASE

Here we prove a result that is used in other parts of this document, namely that if $\mathbf{A} \in \Re^{n \times n}$ is symmetric positive definite and we do the singular value decomposition

$$\mathbf{A} = \mathbf{ULV}^T \tag{234}$$

where by definition $\mathbf{U}$ and $\mathbf{V}$ are orthogonal (e.g. $\mathbf{UU}^T = \mathbf{I}$) and $\mathbf{L}$ diagonal with non-negative diagonal elements, then

$$\mathbf{A} = \mathbf{ULU}^T. \tag{235}$$

This is the same as the spectral decomposition of $\mathbf{A}$ because $\mathbf{U}^T = \mathbf{U}^{-1}$. Equation (235) represents the intersection of the SVD and spectral decompositions of $\mathbf{A}$ since both decompositions are non-unique in different ways. We can prove this result as follows. Firstly, let $\|\cdot\|$ refer to the 2-norm in the rest of this section, i.e. $\|\mathbf{x}\| = \sqrt{\mathbf{x} \cdot \mathbf{x}}$ and $\|\mathbf{M}\| = \max_{\mathbf{x} \neq 0} \frac{\|\mathbf{Mx}\|}{\|\mathbf{x}\|}$. Let $\mathcal{S}$ be the set of vectors $\mathbf{x} \neq 0$ such that $\frac{\|\mathbf{Ax}\|}{\|\mathbf{x}\|} = \|\mathbf{A}\|$. Let the largest singular value of $\mathbf{A}$ (i.e. the largest diagonal element of $\mathbf{L}$) be $\lambda_{\max}$. We can assume that $\lambda_{\max} > 0$ because if it is zero then $\mathbf{U0V}^T = \mathbf{U0U}^T$ and we are done. If k elements of $\mathbf{L}$ share the value $\lambda_{\max}$, with $k \geq 1$, let $i_1 \ldots i_K$ be the set of indices i such that $l_{ii} = \lambda_{\max}$. If $\mathbf{u}_i$ is the i'th column of $\mathbf{U}$ and $\mathbf{v}_i$ the i'th column of $\mathbf{V}$, it is not hard to show from the decomposition of Equation 234 and by symmetry that $\mathcal{S} = \text{span}\,(\mathbf{v}_{i_1} \ldots \mathbf{v}_{i_k}) = \text{span}\,(\mathbf{u}_{i_1} \ldots \mathbf{u}_{i_k})$. $\mathbf{W}$ and $\mathbf{X}$ be the m by k matrices $[\mathbf{u}_{i_1} \ldots \mathbf{u}_{i_k}]$ and $[\mathbf{v}_{i_1} \ldots \mathbf{v}_{i_k}]$ respectively. Because the rows of $\mathbf{W}$ and and $\mathbf{X}$ are orthonormal, we have $\mathbf{W}^T\mathbf{W} = \mathbf{X}^T\mathbf{X} = \mathbf{I}$. Let $\mathbf{w}_1 \ldots \mathbf{w}_k$ be the columns of $\mathbf{W}$ and likewise for $\mathbf{X}$. For any matrix like $\mathbf{W}$ or $\mathbf{X}$ with orthonormal columns (say, $\mathbf{W}$), if $\mathbf{v}$ is in the space spanned by the columns of $\mathbf{W}$ then $\mathbf{WW}^T\mathbf{v} = \mathbf{v}$. This is true for any column of $\mathbf{X}$ as both sets of columns span the same space, so

$$\mathbf{X} = \mathbf{WW}^T\mathbf{X} \tag{236}$$

$$\mathbf{W} = \mathbf{XX}^T\mathbf{W}. \tag{237}$$

Let

$$\mathbf{S} = \mathbf{W}^T\mathbf{X}. \tag{238}$$

Now, $\mathbf{X} = \mathbf{WS}$ from (236) and $\mathbf{W} = \mathbf{XS}^T$ from (237), and multiplying the latter on the right by $\mathbf{S}$ and equating to $\mathbf{X}$ we have $\mathbf{WS} = \mathbf{X} = \mathbf{XS}^T\mathbf{S}$, so $\mathbf{S}^T\mathbf{S}$ is unit (because $\mathbf{X}$ has full rank). This means that $\mathbf{S}$ is orthogonal.

We will now show that $\mathbf{S}$ is symmetric. Suppose that $\mathbf{S}$ is not symmetric, which means we can find some $\mathbf{a}$ such that $\mathbf{Sa} \neq \mathbf{S}^T\mathbf{a}$. We can use this to construct a vector $\mathbf{b}$ such that $\mathbf{Ab} \neq \mathbf{A}^T\mathbf{b}$, which is a contradiction because $\mathbf{A}$ is symmetric. We will set $\mathbf{b} = \mathbf{Xa}$. Because $\mathbf{b}$ is in the span of the columns of $\mathbf{X}$, it is also in the span of the columns of $\mathbf{W}$, and it is orthogonal to all the "other" columns of $\mathbf{U}$ and $\mathbf{V}$, so we can consider only $\mathbf{W}$ and $\mathbf{X}$ when computing $\mathbf{Ab}$. We have $\mathbf{Ab} = \lambda_{\max}\mathbf{WX}^T\mathbf{b} = \lambda_{\max}\left(\mathbf{XS}^T\right)\mathbf{X}^T\left(\mathbf{Xa}\right) = \lambda_{\max}\mathbf{XS}^T\mathbf{a}$. For the transpose, we have $\mathbf{A}^T\mathbf{b} = \lambda_{\max}\mathbf{XW}^T\mathbf{b} = \lambda_{\max}\mathbf{X}\left(\mathbf{SX}^T\right)\left(\mathbf{Xa}\right) = \lambda_{\max}\mathbf{XSa}$. Now, we stated that $\mathbf{Sa} \neq \mathbf{S}^T\mathbf{a}$, and because $\mathbf{X}$ is full rank we can show that $\mathbf{XSa} \neq \mathbf{XS}^T\mathbf{a}$, which means $\mathbf{Ab} \neq \mathbf{A}^T\mathbf{b}$ which is a contradiction.

Because $\mathbf{S}$ is both symmetric and orthogonal, it must be a reflection matrix, and reflection matrices have eigenvalues equal to ± 1. Because $\mathbf{A}$ is positive semi-definite, we can show that $\mathbf{S}$ must not have negative eigenvalues (otherwise, as above we could construct a $\mathbf{b}$ such that $\mathbf{b}^T\mathbf{Ab} < 0$). This means that the eigenvalues of $\mathbf{S}$ must all equal 1, and since it is also symmetric it must be the unit matrix. This means that $\mathbf{W} = \mathbf{X}$.

The rest of the proof is by induction on the number of nonzero singular values in $\mathbf{A}$. We construct

$$\tilde{\mathbf{A}} = \mathbf{A} - \lambda_{\max}\mathbf{WW}^T \tag{239}$$

$$= \mathbf{U}\tilde{\mathbf{L}}\mathbf{V}^T, \tag{240}$$

where $\tilde{\mathbf{L}}$ is as $\mathbf{L}$ but with any diagonal elements equal to $\lambda_{\max}$ set to zero. If we can show that $\tilde{\mathbf{A}} = \mathbf{U}\tilde{\mathbf{L}}\mathbf{U}^T$ we can also show that $\mathbf{A} = \mathbf{ULU}^T$ because we have shown that the singular vectors corresponding to the largest singular values are the same so replacing columns of $\mathbf{V}$ with the corresponding columns of $\mathbf{U}$ would not lead to any change.

C. PRIOR PROBABILITIES FOR STATE VECTORS

This section describes an approach to smoothing the estimates of the vectors $\mathbf{v}_{jkm}$ that is able to take advantage of correlations between the different sub-models (index k). This is a more principled solution to the problem of insufficient data to estimate vectors $\mathbf{v}_{jkm}$ than the *ad-hoc* smoothing approach used in Section 11.1.1. It makes use of a prior distribution over the set of subspace vectors $\mathbf{v}_{jkm}$ for a state j. This becomes a modification to the basic update for the vectors $\mathbf{v}_{jkm}$ of Equation (84). This is an optional feature. The approach described in Section 11.1.1 should be sufficient for a basic implementation.

The use of priors over the vectors $\mathbf{v}_{jkm}$ is mainly motivated by the introduction of sub-models, which makes it likely that we will have instances of vectors $\mathbf{v}_{jk1}$ for particular values of (j, k) that do not have enough training data points to estimate them (the mixture index m would be 1 because we would never split such a vector). Since the vectors $\mathbf{v}_{jkm}$ will not always comprise a majority of the model's parameters we do not anticipate that the use of priors will make a very large difference, unless the number of sub-models K becomes quite large. However it is an attractive feature because it allows us to model correlations between different sub-models (index k) which fixes a weakness of the model.

The way we propose to use priors is to estimate them from the current value of the vectors, on each iteration of training. The prior parameters estimated this way will not be very exact because we have to estimate them from noisy estimates of the vectors, but it is probably better than the *ad-hoc* approach described in Section 11.1.1. A convenient way of modeling the vectors $\mathbf{v}_{jkm}$ is to model the correlations between the mean vector values $\mathbf{v}_{jk}$ defined below:

$$\mathbf{v}_{jk} = \frac{1}{M_{jk}} \sum_{m=1}^{M_{jk}} \mathbf{v}_{jkm}. \tag{241}$$

We model these correlations across different values of k, for a particular j, by concatenating the vectors $\mathbf{v}_{jk}$ into one long vector $\mathbf{v}_j$ of dimension KS and modeling it with a full-covariance Gaussian distribution:

$$\mathbf{v}_j = \begin{bmatrix} \mathbf{v}_{j1} \\ \vdots \\ \mathbf{v}_{jK} \end{bmatrix} \tag{242}$$

$$p(\mathbf{v}_j) = \mathcal{N}(\mathbf{v}_j | \boldsymbol{\mu}^{(\mathrm{pr})}, \boldsymbol{\Sigma}^{(\mathrm{pr})}). \tag{243}$$

We also need to model the deviation of individual vectors $\mathbf{v}_{jkm}$ from the mean $\mathbf{v}_{jk}$ in cases where $M_{jk} > 1$. We derive a suitable model for this in Appendix D, which is:

$$\begin{aligned} p(\mathbf{v}_{jk\cdot} | \mathbf{v}_{jk}) \propto \exp -0.5 \Big(& (M_{jk} - 1)(D \log 2\pi + \log \det \boldsymbol{\Sigma}_k^{(\mathrm{pr})}) \\ & + \sum_{m=1}^{M_{jk}} \mathbf{v}_{jkm}^T \, {\boldsymbol{\Sigma}_k^{(\mathrm{pr})}}^{-1} \mathbf{v}_{jkm} \\ & - M_{jk} \mathbf{v}_{jk} \, {\boldsymbol{\Sigma}_k^{(\mathrm{pr})}}^{-1} \mathbf{v}_{jk} \Big), \end{aligned} \tag{244}$$

with trainable parameters $\boldsymbol{\Sigma}_k^{(\mathrm{pr})}$ for $1 \leq k \leq K$.

C.1. Estimating the prior

The estimation of the prior parameters from an existing set of vectors is quite simple. We first cover the ML estimation and then consider variance flooring. The large matrix $\boldsymbol{\Sigma}^{(\mathrm{pr})}$ is estimated by:

$$\boldsymbol{\mu}^{(\mathrm{pr})} = \frac{1}{J} \sum_{j=1}^{J} \mathbf{v}_j, \tag{245}$$

$$\boldsymbol{\Sigma}^{(\mathrm{pr})} = \left(\frac{1}{J} \sum_{j=1}^{J} \mathbf{v}_j \mathbf{v}_j^T \right) - \boldsymbol{\mu}^{(\mathrm{pr})} {\boldsymbol{\mu}^{(\mathrm{pr})}}^T \tag{246}$$

with $\mathbf{v}_j$ as defined by Equations (241) and (242). Referring to Equation (263) in Appendix D, the prior on the deviations from the means is computed for $1 \leq k \leq K$ as:

$$\boldsymbol{\Sigma}_k^{(\mathrm{pr})} = \frac{\sum_j \left(\sum_{m=1}^{M_{jk}} \mathbf{v}_{jkm} \mathbf{v}_{jkm}^T \right) - M_{jk} \mathbf{v}_{jk} \mathbf{v}_{jk}^T}{\sum_{j=1}^{J} M_{jk} - 1} \tag{247}$$

with $\mathbf{v}_{jk}$ as defined in Equation (241).

C.2. Flooring the prior

Flooring the variance of the prior is necessary for a number of reasons. Firstly, if we try to estimate it from vectors that have just been initialized or whose dimension has just been increased, the prior will have zero variance in at least some dimensions which will stop the training from going anywhere. Secondly, since we will most likely apply the prior during estimation with a scaling factor applied to it, if the scaling factor is too large and we have a lot of parameters with few observations associated with them it is possible to have a situation where the prior can force the parameters to take very small values, which makes the prior even smaller, and so on. A floor on the prior is a good way to arrest this process. Thirdly, it is possible that we may attempt to train a system where the dimension KS of the variance $\boldsymbol{\Sigma}^{(\mathrm{pr})}$ is greater than the number of observations J, and in this case $\boldsymbol{\Sigma}^{(\mathrm{pr})}$ will be singular.

See Appendix I for a description of method used to floor a covariance matrix. It is analogous to diagonal variance flooring except generalized to the full covariance case. There we define a function $\mathbf{A} = \mathrm{floor}(\mathbf{B}, \mathbf{C})$ where $\mathbf{C}$ is the "floor" matrix which must be positive definite and $\mathbf{A}$ is the floored version of $\mathbf{B}$.

The flooring applied to the matrices $\boldsymbol{\Sigma}_k^{(\mathrm{pr})}$ is:

$$\tilde{\boldsymbol{\Sigma}}_k^{(\mathrm{pr})} = \mathrm{floor}\left(\boldsymbol{\Sigma}_k^{(\mathrm{pr})}, \frac{1}{\tau^{(\mathrm{p1})}} \mathbf{H}_k^{(\mathrm{sm})\,-1} \right), \tag{248}$$

with $\mathbf{H}_k^{(\mathrm{sm})}$ as defined in Equation (87). The quantity $\mathbf{H}_k^{(\mathrm{sm})\,-1}$ is dimensionally the same as a variance so it makes sense to use a small multiple of this for the variance floor on the prior. We anticipate using a value of $\tau^{(\mathrm{p1})}$ of around 5 to 10, although the calculation will probably not be very sensitive to this. It will be useful to keep track of how many eigenvalues are floored in the matrix flooring process above; we expect a minority at most to be floored if the process is working as expected. The eigenvalues on the diagonal of $\mathbf{L}$ in Equation (291) will be a useful diagnostic; we expect them to decrease quickly at first and then more slowly.

A suitable formula for flooring the joint variance $\boldsymbol{\Sigma}^{(\mathrm{pr})}$ is:

$$\tilde{\boldsymbol{\Sigma}}^{(\mathrm{pr})} = \mathrm{floor}\left(\boldsymbol{\Sigma}^{(\mathrm{pr})}, \frac{1}{\tau^{(\mathrm{p2})}} \mathbf{F} \right) \tag{249}$$

$$\mathbf{F} = \begin{bmatrix} \mathbf{H}_1^{(\mathrm{sm})\,-1} & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{H}_2^{(\mathrm{sm})\,-1} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}. \tag{250}$$

Again $\tau^{(\mathrm{p2})}$ is a constant that can be set to 5 or 10, and as before we should keep track of the number of eigenvalues floored and inspect the eigenvalues in Equation (291) within the flooring process.

C.3. Applying the prior

This section describes how we apply the priors obtained above during model estimation. We apply them with a weight $\tau^{(\mathrm{pr})}$ which we anticipate might be in the range 5 to 10 for speech tasks. Below, we use the letter $\mathbf{P}$ for the inverse of a variance $\mathbf{\Sigma}$, so for instance $\tilde{\mathbf{P}}_k^{(\mathrm{pr})} = \tilde{\mathbf{\Sigma}}_k^{(\mathrm{pr})\,-1}$. We break up the large precision matrix $\tilde{\mathbf{P}}^{(\mathrm{pr})}$ into S by S blocks and use $\tilde{\mathbf{P}}_{kl}^{(\mathrm{pr})}$ to refer to the block at row-position k, column-position l; similarly we use $\boldsymbol{\mu}_k^{(\mathrm{pr})}$ to refer to the k'th S-dimensional sub-vector of vector $\boldsymbol{\mu}^{(\mathrm{pr})}$. When updating the vector $\mathbf{v}_{jkm}$, the prior term is:

$$\begin{aligned} p(\mathbf{v}_{jkm}) \propto & -0.5\mathbf{v}_{jkm}^T \left(\frac{1}{M_{jk}} \tilde{\mathbf{P}}_{kk}^{(\mathrm{pr})} + \frac{M_{jk}-1}{M_{jk}} \tilde{\mathbf{P}}_{k}^{(\mathrm{pr})} \right) \mathbf{v}_{jkm} \\ & + \mathbf{v}_{jkm}^T \left(\sum_{m' \in \{1 \ldots M_{jk}\} \setminus \{m\}} \frac{1}{M_{jk}} \tilde{\mathbf{P}}_{k}^{(\mathrm{pr})} \mathbf{v}_{jkm'} \right. \\ & \left. - \sum_{k'=1}^{K} \tilde{\mathbf{P}}_{kk'}^{(\mathrm{pr})} \left((1-\delta(k',k)) \mathbf{v}_{jk'} - \boldsymbol{\mu}_{k'}^{(\mathrm{pr})} \right) \right). \end{aligned} \quad (251)$$

We would then use modified values $\mathbf{H}_{jkm}$ and $\mathbf{g}_{jkm}$ when computing the vector $\mathbf{v}_{jkm}$ using Equation (84), as follows:

$$\tilde{\mathbf{H}}_{jkm} = \mathbf{H}_{jkm} + \tau^{(\mathrm{pr})} \left(\frac{1}{M_{jk}} \tilde{\mathbf{P}}_{kk}^{(\mathrm{pr})} + \frac{M_{jk}-1}{M_{jk}} \tilde{\mathbf{P}}_{k}^{(\mathrm{pr})} \right) \quad (252)$$

$$\begin{aligned} \tilde{\mathbf{g}}_{jkm} = & \mathbf{g}_{jkm} + \tau^{(\mathrm{pr})} \left(\sum_{m' \in \{1 \ldots M_{jk}\} \setminus \{m\}} \frac{1}{M_{jk}} \tilde{\mathbf{P}}_{k}^{(\mathrm{pr})} \hat{\mathbf{v}}_{jkm'} \right. \\ & \left. - \sum_{k'=1}^{K} \tilde{\mathbf{P}}_{kk'}^{(\mathrm{pr})} \left((1-\delta(k',k)) \hat{\mathbf{v}}_{jk'} - \boldsymbol{\mu}_{k'}^{(\mathrm{pr})} \right) \right) \end{aligned} \quad (253)$$

$$\hat{\mathbf{v}}_{jkm} = \tilde{\mathbf{H}}_{jkm}^{-1} \tilde{\mathbf{g}}_{jkm} \quad (254)$$

Note that the vectors that appear on the right of Equations (252) have a hat on. This is supposed to indicate that where other vectors within the same state appear in the equation, we should use the already updated versions if they have already been computed, rather than the pre-update ones. The vector which is being updated, $\mathbf{v}_{jkm}$, appears with zero coefficient on the right hand side of Equation (253) so Equation (254) does not contain a circular reference. This update may be done several times for each state, iterating over k and m each time, to get a more exact answer. The reason why we formulate it like this is to avoid the need to invert a very large matrix for each state.

The data likelihood improvement should be measured using the unmodified form of Equation (81). With a prior involved we can no longer guarantee that the auxiliary function excluding the prior term will improve on each iteration but we still expect it to improve in practice.

D. MODELING OFFSETS FROM A MEAN

For the prior distributions over the vectors $\mathbf{v}_{jkm}$, we require a model for deviations from a mean value in a particular situation. The situation is, suppose we have a collection of N vectors $\mathbf{x}_1 \ldots \mathbf{x}_N$ and we have already somehow modeled their mean $\bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n$. We want to model the deviations from the mean. These deviations $\hat{\mathbf{x}}_n \equiv \mathbf{x}_n - \bar{\mathbf{x}}$ cannot be modeled independently because they are correlated (they are constrained to sum to one). A sensible probability model is to assume that the vectors $\mathbf{x}_n$ were independently generated with a variance $\mathbf{\Sigma}$, and the mean was then removed. We use this assumption below to work out the distribution of the offset vectors $\hat{\mathbf{x}}_n$.

Let us first cover the single-dimensional scalar case with $\mathbf{\Sigma} = [1]$ and then generalize to higher dimensions. We retain the vector notation even though the vectors have only one dimension. The distribution of $\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_N \end{bmatrix}$ (before normalization) is $\mathbf{I}_N$. We now want to work out the probability distribution over the offsets $\hat{\mathbf{x}}_n$. We can use a form of Bayes' rule:

$$p(\hat{\mathbf{x}}|\bar{\mathbf{x}}) = \frac{p(\hat{\mathbf{x}}, \bar{\mathbf{x}})}{p(\bar{\mathbf{x}})} \quad (255)$$

$$= \frac{Kp(\mathbf{x})}{p(\bar{\mathbf{x}})}, \quad (256)$$

with K a constant that reflects that fact that although $\hat{\mathbf{x}}$ and $\bar{\mathbf{x}}$ determine $\mathbf{x}$ and vice versa, the likelihood values $p(\hat{\mathbf{x}}, \bar{\mathbf{x}})$ and $p(\mathbf{x})$ may not be the same because of scaling issues. This may depend on a choice of what exactly we mean when we write $p(\hat{\mathbf{x}})$, but for current purposes it does not matter. K is a function only of N and we will not need to work out its value. We can work out $p(\bar{\mathbf{x}})$ by noting that it has variance $\frac{1}{N}$, and using the fact that $\bar{\mathbf{x}} = \frac{1}{N}\mathbf{x}^T\mathbf{1}$ with $\mathbf{1}$ a vector of all ones. Ignoring normalizers, the distributions over $\mathbf{x}$, $\bar{\mathbf{x}}$ and $\hat{\mathbf{x}}$ can be expressed as:

$$p(\mathbf{x}) \propto \exp\left(-0.5\mathbf{x}^T\mathbf{I}\mathbf{x}\right) \quad (257)$$

$$p(\bar{\mathbf{x}}) \propto \exp\left(\frac{-0.5}{N}\mathbf{x}^T\mathbf{1}\mathbf{1}^T\mathbf{x}\right) \quad (258)$$

$$p(\hat{\mathbf{x}}|\bar{\mathbf{x}}) \propto \exp\left(-0.5\mathbf{x}^T\left(\mathbf{I} - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right)\mathbf{x}\right). \quad (259)$$

The normalizing factor for Equation (259) can be worked out from Equation (256), putting in $\mathbf{x} = 0$. We get:

$$p(\hat{\mathbf{x}}|\bar{\mathbf{x}}) = K\exp -\frac{1}{2}\left((N-1)\log 2\pi\right. \quad (260)$$

$$\left. + \mathbf{x}^T\left(\mathbf{I} - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right)\mathbf{x}\right. \quad (261)$$

Note that although we write the likelihood as $p(\hat{\mathbf{x}}|\bar{\mathbf{x}})$ the result does not actually depend on the value of $\bar{\mathbf{x}}$. For the vector-valued case with non-unit variance $\mathbf{\Sigma}$ of dimension D, we can work out:

$$\begin{aligned} p(\hat{\mathbf{x}}|\bar{\mathbf{x}}) = & K\exp -\frac{1}{2}\left((N-1)(D\log 2\pi + \log\det\mathbf{\Sigma})\right. \\ & \left. + \sum_{n=1}^{N} \mathbf{x}_n^T\mathbf{\Sigma}^{-1}\mathbf{x}_n - N\bar{\mathbf{x}}\mathbf{\Sigma}^{-1}\bar{\mathbf{x}}\right). \end{aligned} \quad (262)$$

If we are in a situation where we want to train the variance $\mathbf{\Sigma}$ given given a collection of sets $\mathbf{x}_1^{(m)} \ldots \mathbf{x}_{N_m}^{(m)}$ for $1 \leq m \leq M$, each potentially of a different size N_m, the update equation is:

$$\mathbf{\Sigma} = \frac{\sum_{m=1}^{M}\left(\sum_{n=1}^{N_m} \mathbf{x}_n^{(m)}\ \mathbf{x}_n^{(m)\,T}\right) - N_m\bar{\mathbf{x}}^{(m)}\ \bar{\mathbf{x}}^{(m)\,T}}{\sum_{m=1}^{M} N_m - 1} \quad (263)$$

with the average quantities $\bar{\mathbf{x}}^{(m)}$ being defined in the obvious way. The derivation is fairly simple.

E. MAXIMUM LIKELIHOOD GAUSSIAN CLUSTERING ALGORITHM

What we are describing in this section is an algorithm to do Maximum Likelihood clustering of J diagonal covariance Gaussians to a smaller number I of diagonal covariance Gaussians, by minimizing the likelihood loss (weighted by w_j) that we would get if we were to model the data from each Gaussian $1 \leq j \leq J$ by the Gaussian from its cluster $1 \leq i \leq I$. Note that the indices j and i as used here have no intrinsic connection with the same letters used elsewhere in this document (except that the number of clusters I will typically be the same as the number of Gaussians I in the shared GMM).

The input to this algorithm is a set of J Gaussians with weights w_j, means $\boldsymbol{\mu}_j$ and diagonal covariances Σ_j (with diagonal elements σ_{jd}^2). The output is a set of I Gaussians corresponding to cluster centers, with weights $\bar{w}_i$, means $\bar{\boldsymbol{\mu}}_i$ and diagonal variances $\bar{\Sigma}_i$, together with a mapping from each of the J Gaussians to the I cluster centers which are represented as diagonal Gaussian distributions.

This algorithm is not particularly fast. There are various ways of speeding it up but they tend be quite complicated. We suggest initially throwing away all but the Gaussians with the highest counts (e.g. just keep the 10k most likely Gaussians) in order to make it acceptably fast, and also limiting the number of iterations, e.g. to 40 or so. Regardless, Gaussians with zero weights should be thrown away at the start. We should also avoid taking a log on each dimension of the computations below, instead keeping a running product and taking a log at the end; we can detect if we have a floating point overflow or underflow by checking for zeros or infinities when we come to take the log, and in that case just back off to the inefficient version.

The algorithm is easiest to write down if we represent the initial and clustered Gaussians as statistics. We will write

$$
\begin{aligned}
c_j &= w_j && (264)\\
\mathbf{m}_j &= c_j \boldsymbol{\mu}_j && (265)\\
\mathbf{s}_j &= s_{jd}, 1 \leq d \leq D: && (266)\\
s_{jd} &= c_j(\mu_{jd}^2 + \sigma_{jd}^2) && (267)
\end{aligned}
$$

as the zeroth, first and diagonal second order statistics respectively of the initial Gaussians. Let us use the notation $\mathcal{S}_i = \{j_1, j_2 \ldots\}$ as the set of Gaussians in cluster i, and $c(j)$ as the cluster to which j currently belongs. We start with a random assignment of Gaussians to clusters, e.g. use $c(j) = (j \bmod I) + 1$. We maintain throughout the algorithm the statistics for the clustered Gaussians, which are always equal to:

$$
\begin{aligned}
\bar{c}_i &= \sum_{j \in \mathcal{S}_i} c_j && (268)\\
\bar{\mathbf{m}}_i &= \sum_{j \in \mathcal{S}_i} \mathbf{m}_j && (269)\\
\bar{\mathbf{s}}_i &= \sum_{j \in \mathcal{S}_i} \mathbf{s}_j && (270)
\end{aligned}
$$

The likelihood contribution of a cluster i is:

$$
l(i) = -0.5\bar{c}_i \left(2\pi D + D + \log \prod_{d=1}^{D} \left(\frac{\bar{s}_{id}}{\bar{c}_i} - \frac{\bar{\mathbf{m}}_{id}^2}{\bar{c}_i^2} \right) \right), \quad (271)
$$

and we can ignore the $2\pi D + D$ as it will not affect the answer. The objective function we are optimizing is the sum of all the $l(i)$. It is useful to keep the values of $l(i)$ stored throughout the process. The basic operation of this algorithm is: supposing Gaussian j is currently in cluster i (and is not the only element in cluster i), try moving j to some other cluster i' and see if this would increase the likelihood. We do this by making a temporary copy of the statistics for states i and i', subtracting the statistics for j from i and adding them to i', and computing the altered values of $l(i)$ and $l(i')$. If the sum of the altered values $l(i) + l(i')$ is greater than the current sum, we can move j from cluster i to i'. Moving j will involve keeping various quantities updated: $S_i, \bar{c}_i, \bar{\mathbf{m}}_i, \bar{\mathbf{s}}_i, l(i), S_{i'}, \bar{c}_{i'}, \bar{\mathbf{m}}_{i'}, \bar{\mathbf{s}}_{i'}, l(i'), c(j)$.

The exact order of attempting to move Gaussian j from i to i' is up to the coder. The obvious approach is: on each iteration we visit each j which is not part of a singleton cluster, and actually test the improvement we get from moving j to each i' other than $c(j)$, and then pick the best. A possibly more efficient approach would be on each iteration to test moving j to some subset of all the i', where the subset is determined by the iteration number: e.g. on iteration p, pick all i' such that $(i' + j) \equiv p (\bmod K)$ for some K (e.g. $K = 10$). Then the first time we find an i' that we would be willing to move j to, move it and continue on to the next j.

The stopping criterion can be either to stop when we see no more changes (e.g. when we have gone K iterations with no Gaussians moving), or to stop after a predetermined number of iterations. After we stop, we have to convert the statistics back into the form of a mean and variance and a weight. This is quite obvious:

$$
\begin{aligned}
\bar{\boldsymbol{\mu}}_i &= \frac{1}{\bar{c}_i} \bar{\mathbf{m}}_i && (272)\\
\bar{\sigma}_{id}^2 &= \frac{\bar{s}_{id}}{\bar{c}_i} - \frac{\bar{\mathbf{m}}_{id}^2}{\bar{c}_i^2} && (273)\\
\bar{w}_i &= \frac{\bar{c}_i}{\sum_i \bar{c}_i}. && (274)
\end{aligned}
$$

The covariances $\bar{\boldsymbol{\Sigma}}_i$ that result from this procedure are diagonal.

F. MATRIX CALCULUS DERIVATIONS

F.1. Derivations for Equations (217) and (218)

For Equations (217) and (218) we need to compute the first and second derivatives of the expression $\log\det(\mathbf{A} + k\boldsymbol{\Delta})$ with respect to k. The first derivative is

$$
\frac{\mathbf{d}}{\mathbf{d}k} \log\det(\mathbf{A} + k\boldsymbol{\Delta}) = \mathrm{tr}\left((\mathbf{A} + k\boldsymbol{\Delta})^{-1}\boldsymbol{\Delta}\right), \quad (275)
$$

which we can obtain from the formula $\frac{\mathbf{d}}{\mathbf{d}x} \log|\mathbf{M}| = \mathrm{tr}\left(\mathbf{M}^{-1} \frac{\mathbf{dM}}{\mathbf{d}x}\right)$. For the second derivative we first use $\frac{\mathbf{d}}{\mathbf{d}x}\mathrm{tr}\left(\mathbf{M}\right) = \mathrm{tr}\left(\frac{\mathbf{d}M}{\mathbf{d}x}\right)$. This tells us that

$$
\frac{\mathbf{d}^2}{\mathbf{d}k^2} \log\det(\mathbf{A} + k\boldsymbol{\Delta}) = \mathrm{tr}\left(\frac{\mathbf{d}}{\mathbf{d}x}((\mathbf{A} + k\boldsymbol{\Delta})^{-1}\boldsymbol{\Delta})\right). \quad (276)
$$

We then use the formula $\frac{\mathbf{d}}{\mathbf{d}x}(\mathbf{MN}) = \mathbf{M}\frac{\mathbf{d}N}{\mathbf{d}x} + \frac{\mathbf{dM}}{\mathbf{d}x}\mathbf{N}$ to turn the right hand side of Equation (276) to $\mathrm{tr}\left((\frac{\mathbf{d}}{\mathbf{d}x}(\mathbf{A} + k\boldsymbol{\Delta})^{-1})\boldsymbol{\Delta}\right)$. We can use the formula $\frac{\mathbf{d}}{\mathbf{d}x}\mathbf{M}^{-1} = -\mathbf{M}^{-1}\frac{\mathbf{dM}}{\mathbf{d}x}\mathbf{M}^{-1}$ with $\mathbf{M}$ equivalent to $(\mathbf{A} + k\boldsymbol{\Delta})$, to arrive at the final formula:

$$
\frac{\mathbf{d}^2}{\mathbf{d}k^2} \log\det(\mathbf{A} + k\boldsymbol{\Delta}) = \mathrm{tr}\left((\mathbf{A} + k\boldsymbol{\Delta})^{-1}\boldsymbol{\Delta}(\mathbf{A} + k\boldsymbol{\Delta})^{-1}\boldsymbol{\Delta}\right). \quad (277)
$$

G. OPTIMIZING POORLY CONDITIONED QUADRATIC AUXILIARY FUNCTIONS

G.1. Vector optimizations

The techniques described in this document include a number of problems where the auxiliary function and update are of the general form:

$$\mathcal{Q}(\mathbf{x}) = \mathbf{x} \cdot \mathbf{g} - \frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} \quad (278)$$

$$\hat{\mathbf{x}} = \mathbf{H}^{-1}\mathbf{g}, \quad (279)$$

where $\mathbf{H}$ is symmetric. There is no problem here if $\mathbf{H}$ is of full rank; however, there are many cases we encounter where either $\mathbf{H}$ is not of full rank, or the condition of $\mathbf{H}$ (the ratio of smallest to largest singular values) is so large that $\mathbf{H}$ is indistinguishable from a reduced rank matrix. This arises naturally where $\mathbf{H}$ is a weighted outer product of vectors $\mathbf{v}_n$ and $\mathbf{g}$ is a weighted sum of $\mathbf{v}_n$:

$$\mathbf{H} = \sum_{n=1}^{N} a_n \mathbf{v}_n \mathbf{v}_n^T \quad (280)$$

$$\mathbf{g} = \sum_{n=1}^{N} b_n \mathbf{v}_n, \quad (281)$$

with $\mathbf{a}_n \geq 0$. It will always be the case in the problems we deal with here that nonzero b_n implies nonzero a_n (otherwise the auxiliary function might have an infinite maximum value). If the number of nonzero a_n is smaller than the dimension of the problem, $\mathbf{A}$ will be of reduced rank and we cannot invert it. If we had access to the original vectors $\mathbf{v}_n$ we could solve the problem in a least squares sense in the space spanned by the vectors, but we are generally forced to work from the statistics $\mathbf{g}$ and $\mathbf{H}$ themselves. Numerically determining the rank of a matrix like $\mathbf{H}$ is usually not practical [12]. Therefore we have developed a procedure for solving this problem which is robust given imprecise statistics. Because we cannot exactly identify the "null-space" of the problem, trying to solve the problem in a least squares sense and setting those dimensions of $\mathbf{x}$ to zero would be dangerous. Instead we aim to leave $\mathbf{x}$ the same as it originally was in those dimensions by reformulating the problem in terms of the offset $\Delta = \hat{\mathbf{x}} - \mathbf{x}$.

The method is as follows:

- If $\mathbf{H}$ is the zero matrix, leave the variable $\mathbf{x}$ the same. Otherwise:
- Compute $\bar{\mathbf{g}} = \mathbf{g} - \mathbf{H}\mathbf{x}$. This is the value of $\mathbf{g}$ in the auxiliary function in Δ.
- Do the SVD $\mathbf{H} = \mathbf{U}\mathbf{L}\mathbf{V}^T$, which implies $\mathbf{H} = \mathbf{U}\mathbf{L}\mathbf{U}^T$ because $\mathbf{H}$ is positive semi-definite.
- Compute a floor $f = \max(\epsilon, \frac{\max_i l_{ii}}{K})$, with K a maximum condition (e.g. 10^4) and ϵ for example 10^{-40}.
- Compute the floored diagonal matrix $\tilde{\mathbf{L}}$, with $\tilde{l}_{ii} = \max(f, l_{ii})$.
- Compute $\Delta = \mathbf{U}(\tilde{\mathbf{L}}^{-1}(\mathbf{U}^T\bar{\mathbf{g}}))$ (the bracketing shows the order of evaluation).
- Compute $\hat{\mathbf{x}} = \mathbf{x} + \Delta$.
- Measure the change in the auxiliary function of Equation 278 between $\mathbf{x}$ and $\hat{\mathbf{x}}$.
- If this change is negative, print out a warning, do not update this parameter (return $\mathbf{x}$), and continue. Otherwise, return $\hat{\mathbf{x}}$ and accumulate the total auxiliary function change for diagnostic purposes.

Note that if the largest (absolute) element of $\mathbf{H}$ is nonzero but very tiny (e.g. less than 10^{-40}), it may be necessary to scale it before doing the SVD and then apply the reverse scale to the matrix $\mathbf{L}$; standard SVD algorithms can fail with very small values.

G.2. Matrix optimizations

In the update of the projection matrices $\mathbf{M}_{ki}$ and $\mathbf{N}_{ki}$ we encounter an auxiliary function and update which is of the general form:

$$\mathcal{Q}(\mathbf{M}) = \mathrm{tr}\,(\mathbf{M}^T\mathbf{\Sigma}^{-1}\mathbf{Y}) - \frac{1}{2}\mathrm{tr}\,(\mathbf{\Sigma}^{-1}\mathbf{M}\mathbf{Q}\mathbf{M}^T) \quad (282)$$

$$\hat{\mathbf{M}} = \mathbf{Y}\mathbf{Q}^{-1}, \quad (283)$$

with $\mathbf{Q}$ a symmetric positive semi-definite matrix, and $\mathbf{Y}$ the same dimension as $\mathbf{M}$, and $\mathbf{\Sigma}$ is symmetric positive definite. Again the problem arises if the condition of $\mathbf{Q}$ is poor. We will simply state the procedure used, as the rationale is the same as above.

- If $\mathbf{Q}$ is the zero matrix, do not update $\mathbf{M}$. Otherwise:
- Compute $\bar{\mathbf{Y}} = \mathbf{Y} - \mathbf{M}\mathbf{Q}$
- Do the SVD $\mathbf{Q} = \mathbf{U}\mathbf{L}\mathbf{V}^T$, which implies $\mathbf{Q} = \mathbf{U}\mathbf{L}\mathbf{U}^T$ because $\mathbf{Q}$ is positive semi-definite.
- Compute a floor $f = \max(\epsilon, \frac{\max_i l_{ii}}{K})$, with K a maximum condition (e.g. 10^4) and ϵ for example 10^{-40}.
- Compute the floored diagonal matrix $\tilde{\mathbf{L}}$, with $\tilde{l}_{ii} = \max(f, l_{ii})$.
- Compute $\Delta = ((\bar{\mathbf{Y}}\mathbf{U})\tilde{\mathbf{L}}^{-1})\mathbf{U}^T$, and $\hat{\mathbf{M}} = \mathbf{M} + \Delta$.
- Compute the change in auxiliary function by evaluating Equation (282) for $\mathbf{M}$ and $\hat{\mathbf{M}}$.
- If the auxiliary function decreased, print a warning, return the old value $\mathbf{M}$ and continue. Otherwise, accumulate the change in auxiliary function for diagnostic purposes and return the new value $\hat{\mathbf{M}}$.

H. CONDITION-LIMITED INVERSION

Here we describe a method of inverting symmetric positive semi-definite matrices using a method that gives a result for matrices that are singular or close to singular. Given a symmetric positive semi-definite matrix $\mathbf{A}$, we are returning $\tilde{\mathbf{A}}^{-1}$, where $\tilde{\mathbf{A}}$ is a matrix very close to $\mathbf{A}$ but modified to floor its eigenvalues the largest eigenvalue of $\mathbf{A}$ divided by a specified condition number K.

The 2-norm condition $\kappa_2(\mathbf{A})$ is the ratio of largest to smallest singular values (it can be defined in various ways depending on the matrix norm used, but this is a common one). For matrix inversion to be stable, the condition should be much smaller than the inverse of the machine precision [12], e.g. much smaller than about 2^{24} for single precision arithmetic. Typically, depending on the task, we will be able to limit the condition to be much smaller than this, e.g. to 1000 or so.

First we define a function:

$$\tilde{\mathbf{A}} = \mathrm{limitcond}(\mathbf{A}, K), \quad (284)$$

which limits the condition of a symmetric positive semi-definite matrix $\mathbf{A}$ by flooring its eigenvalues to $1/K$ times the largest eigenvalue. The process of computing $\tilde{\mathbf{A}}$ gives an easy way to compute its inverse.

Given a matrix $\mathbf{A}$ which is symmetric positive semi-definite, and a specified maximum condition K, we do the singular value decomposition

$$\mathbf{A} = \mathbf{U}\mathbf{L}\mathbf{V}^T \tag{285}$$

with $\mathbf{U}$ and $\mathbf{V}$ orthogonal and $\mathbf{L}$ diagonal, and return the matrix

$$\tilde{\mathbf{A}} = \mathbf{U}\tilde{\mathbf{L}}\mathbf{U}^T \tag{286}$$

with the modified diagonal matrix $\tilde{\mathbf{L}}$ with its diagonal elements floored to the value $f = \max(\epsilon, \frac{\max_i l_{ii}}{k})$, so $\tilde{l}_{ii} = \max(f, l_{ii})$. We can use a number like 10^{-40} for ϵ; this is to avoid getting infinite answers. In the kinds of problems where we will use this, this is acceptable. The number of diagonal elements floored is a useful diagnostic. In the situations where condition-limiting is useful we will generally actually require the inverse of this condition-limited matrix, which we can do as:

$$\tilde{\mathbf{A}}^{-1} = \mathbf{U}\tilde{\mathbf{L}}^{-1}\mathbf{U}^T. \tag{287}$$

Assuming $\mathbf{A}$ was positive semi-definite, the difference $\|\tilde{\mathbf{A}} - \mathbf{A}\|_2$ will be no more than f. Note that we return $\mathbf{U}^T$ rather than $\mathbf{V}^T$ on the right to ensure symmetry and positive definiteness– if $\mathbf{A}$ is symmetric, $\mathbf{U}$ and $\mathbf{V}$ will in general be the same up to signs of the columns, and these signs will be the same for positive definite $\mathbf{A}$. However if $\mathbf{A}$ has a rank deficiency of more than two, the rows and columns of $\mathbf{U}$ and $\mathbf{V}$ corresponding to the null-space could be rotated arbitrarily, and by replacing $\mathbf{V}$ with $\mathbf{U}$ we ensure that the result is symmetric. Note that if a column of $\mathbf{V}$ had a different sign from the corresponding column of $\mathbf{U}$ and the corresponding l_{ii} was larger than the floor we would effectively be flipping the sign of the eigenvalue from negative to positive, but this violates our assumption that $\mathbf{A}$ was initially positive semi-definite.

Note that if the largest absolute value of any element in $\mathbf{A}$ is nonzero but very tiny, e.g. less than 10^{-40}, standard singular value decomposition algorithms may fail. Our implementation of singular value decomposition detects this condition and prescales the matrix before giving it to the standard algorithms, and then scales the singular values afterward by the inverse of the pre-scaling factor.

I. FLOORING A MATRIX

Here we describe a general procedure for flooring a matrix, that is useful for flooring full covariance matrices and for other purposes. We define the function

$$\mathbf{A} = \text{floor}(\mathbf{B}, \mathbf{C}) \tag{288}$$

where $\mathbf{B}$ is a symmetric positive semi-definite matrix and $\mathbf{C}$ is a symmetric positive definite matrix of the same dimension, as follows. First, we do the Cholesky decomposition $\mathbf{C} = \mathbf{L}\mathbf{L}^T$. Then we define

$$\mathbf{D} = \mathbf{L}^{-1}\mathbf{B}\mathbf{L}^{-T} \tag{289}$$

We do the singular value decomposition

$$\mathbf{D} = \mathbf{U}\mathbf{M}\mathbf{V}^T \tag{290}$$

which because $\mathbf{D}$ is symmetric positive semi-definite implies

$$\mathbf{D} = \mathbf{U}\mathbf{M}\mathbf{U}^T \tag{291}$$

(see Appendix B), with $\mathbf{M}$ diagonal and $\mathbf{U}$ orthogonal, and define $\mathbf{M}'$ as the diagonal matrix $\mathbf{M}$ of Equation (291) with its diagonal elements floored at 1, so $m'_{ii} = \max(1, m_{ii})$. Then we set

$$\mathbf{D}' = \mathbf{U}\mathbf{M}'\mathbf{U}^T \tag{292}$$

$$\mathbf{A} = \mathbf{L}\mathbf{D}'\mathbf{L}^T. \tag{293}$$

This function $\text{floor}(\cdot,\cdot)$ is like a $\max$ operation on symmetric matrices, except that it treats the first and second arguments differently (in particular, the second is required to be positive definite).

J. ESTIMATING AND USING PRIORS OVER THE PROJECTIONS

In this section we describe how to train and use prior distributions over the projections $\mathbf{M}$ and $\mathbf{N}$. This is in order to get better parameter estimates and to solve the problems of poor conditioning that we encounter during update.

We will describe the procedure only for $\mathbf{M}$ since $\mathbf{M}$ and $\mathbf{N}$ are mirror images of each other and the generalization is obvious. The prior distribution we will use over $\mathbf{M}$ is a Gaussian prior with a constrained covariance structure. Essentially we are trying to find a covariance $\mathbf{\Sigma}_r$ and $\mathbf{\Sigma}_l$ such that the elements of $\mathbf{\Sigma}_l^{-0.5}\mathbf{M}\mathbf{\Sigma}_r^{-0.5}$ are independently distributed with unit variance. We will assume that we are estimating a single prior that is shared between all submodels k; we do this because it is possible that the number of projections I_k within a particular k could be less than the number of rows or columns of $\mathbf{M}$, which would cause problems in parameter estimation. This procedure does assume that the total number of projections is more than the larger of the number of rows or columns of $\mathbf{M}$, but if not we could simply limit the condition of the covariances by some floor as described in Appendix H.

We will simply state the procedure as the derivation is not too difficult. Recall that $\mathbf{M}_{ki}$ is a D by $S{+}1$ matrix. First we find the mean $\bar{\mathbf{M}} = \frac{1}{\sum_k I_k}\sum_{k,i}\mathbf{M}_{ki}$. We initialize $\mathbf{\Sigma}_r = \mathbf{I} \in \Re^{(S+1)\times(S+1)}$ and $\mathbf{\Sigma}_l = \mathbf{I} \in \Re^{D\times D}$. Then for several iterations (e.g. five iterations), we do:

$$\mathbf{\Sigma}_l = \frac{1}{D\sum_k I_k}\sum_{k,i}(\mathbf{M}_{ki}-\bar{\mathbf{M}})\mathbf{\Sigma}_r^{-1}(\mathbf{M}_{ki}-\bar{\mathbf{M}})^T \tag{294}$$

$$\mathbf{\Sigma}_r = \frac{1}{(S+1)\sum_k I_k}\sum_{k,i}(\mathbf{M}_{ki}-\bar{\mathbf{M}})^T\mathbf{\Sigma}_l^{-1}(\mathbf{M}_{ki}-\bar{\mathbf{M}}). \tag{295}$$

At this point we can limit the condition of $\mathbf{\Sigma}_l$ and $\mathbf{\Sigma}_r$ to some large value (say, 1000) by flooring eigenvalues as described in Appendix H, in order to cover the case where there are too few matrices to estimate the prior. We can make sense of these equations by noticing that if $\tilde{\mathbf{M}}_{ki} = (\mathbf{M}_{ki} - \bar{\mathbf{M}})\mathbf{\Sigma}_r^{-0.5}$ which is $\mathbf{M}_{ki}$ with the global mean removed and with the columns decorrelated, and if $\tilde{\mathbf{m}}_{kid}$ is the d'th row of this, then we are setting $\mathbf{\Sigma}_l$ to the variance of these rows. The expression for the likelihood of a matrix $\mathbf{M}$ given our prior is:

$$\log P(\mathbf{M}) = \exp -\frac{1}{2}\Big(D(S{+}1)\log(2\pi) + (S{+}1)\det\mathbf{\Sigma}_r + D\det\mathbf{\Sigma}_l + \text{tr}\left(\mathbf{\Sigma}_l^{-1}(\mathbf{M}-\bar{\mathbf{M}})\mathbf{\Sigma}_r^{-1}(\mathbf{M}-\bar{\mathbf{M}})^T\right)\Big). \tag{296}$$

We will scale the prior with a scale $\tau^{(\text{pr})}$ (e.g. 10 or 20), the same value that we use for the priors over the other parameters. This is related to the language model scale used in decoding as it is the weighting factor between probabilities produced by our model and "real"

probabilities. The auxiliary function $\mathcal{Q}'(\mathbf{M}_{ki})$ that includes $\tau^{(\mathrm{pr})}$ times the prior plus the original auxiliary function of Equation (100) can be written as follows:

$$\begin{aligned}\mathcal{Q}'(\mathbf{M}_{ki}) &= K'' + \mathrm{tr}\left(\mathbf{M}^T\boldsymbol{\Sigma}_{ki}^{-1}\mathbf{Y}_{ki}\right) \\ &\quad -\frac{1}{2}\mathrm{tr}\left(\boldsymbol{\Sigma}_{ki}^{-1}\mathbf{M}_{ki}\mathbf{Q}_{ki}\mathbf{M}_{ki}^T\right) \\ &\quad +\tau^{(\mathrm{pr})}\mathrm{tr}\left(\mathbf{M}^T\boldsymbol{\Sigma}_l^{-1}\bar{\mathbf{M}}\boldsymbol{\Sigma}_r^{-1}\right) \\ &\quad -\frac{\tau^{(\mathrm{pr})}}{2}\mathrm{tr}\left(\boldsymbol{\Sigma}_l^{-1}\mathbf{M}_{ki}\boldsymbol{\Sigma}_r^{-1}\mathbf{M}_{ki}^T\right). \quad (297)\end{aligned}$$

We can write this more compactly as:

$$\begin{aligned}F(\mathbf{M}) &= \mathrm{tr}\left(\mathbf{M}^T\mathbf{G}\right) - \frac{1}{2}\mathrm{tr}\left(\mathbf{P}_1\mathbf{M}\mathbf{Q}_1\mathbf{M}^T\right) \\ &\quad -\frac{1}{2}\mathrm{tr}\left(\mathbf{P}_2\mathbf{M}\mathbf{Q}_2\mathbf{M}^T\right) \quad (298)\\ \mathbf{G} &= \boldsymbol{\Sigma}_{ki}^{-1}\mathbf{Y}_{ki} + \tau^{(\mathrm{pr})}\boldsymbol{\Sigma}_l^{-1}\bar{\mathbf{M}}\boldsymbol{\Sigma}_r^{-1} \quad (299)\\ \mathbf{P}_1 &= \boldsymbol{\Sigma}_{ki}^{-1} \quad (300)\\ \mathbf{Q}_1 &= \mathbf{Q}_{ki} \quad (301)\\ \mathbf{P}_2 &= \boldsymbol{\Sigma}_l^{-1} \quad (302)\\ \mathbf{Q}_2 &= \tau^{(\mathrm{pr})}\boldsymbol{\Sigma}_r^{-1}, \quad (303)\end{aligned}$$

where $\mathbf{P}_1$, $\mathbf{P}_2$ and $\mathbf{Q}_2$ will be symmetric positive definite and $\mathbf{Q}_1$ will be symmetric positive semidefinite. We can maximize Equation (298) by computing a transformation $\mathbf{T}$ that simultaneously diagonalizes $\mathbf{P}_1$ and $\mathbf{P}_2$ and doing a row by row update in the transformed space. Let us suppose we want to make $\mathbf{P}_1$ unit and diagonalize $\mathbf{P}_2$. We do the Cholesky decomposition

$$\mathbf{P}_1 = \mathbf{L}\mathbf{L}^T, \quad (304)$$

define $\mathbf{S} = \mathbf{L}^{-1}\mathbf{P}_2\mathbf{L}^{-T}$, do the SVD

$$\mathbf{S} = \mathbf{U}\mathbf{D}\mathbf{V}^T \quad (305)$$

(which implies $\mathbf{S} = \mathbf{U}\mathbf{D}\mathbf{U}^T$ because $\mathbf{S}$ is positive definite, see Appendix B), and our transform is

$$\mathbf{T} = \mathbf{U}^T\mathbf{L}^{-1}, \quad (306)$$

so that $\mathbf{T}\mathbf{P}_1\mathbf{T}^T = \mathbf{I}$ and $\mathbf{T}\mathbf{P}_2\mathbf{T}^T = \mathbf{D}$. Let us define $\mathbf{M}' = \mathbf{T}^{-T}\mathbf{M}$, and if we define $\mathbf{G}' = \mathbf{T}\mathbf{G}$ we can write our auxiliary function as:

$$\begin{aligned}F(\mathbf{M}') &= \mathbf{M}'^T\mathbf{G}' - \frac{1}{2}\mathrm{tr}\left(\mathbf{M}'\mathbf{Q}_1\mathbf{M}'^T\right) \\ &\quad -\frac{1}{2}\mathrm{tr}\left(\mathbf{D}\mathbf{M}'\mathbf{Q}_2\mathbf{M}'^T\right). \quad (307)\\ & \quad (308)\end{aligned}$$

We can separate this out for each row $\mathbf{m}_n$ of $\mathbf{M}'$, so (using $\mathbf{g}'_n$ as the n'th row of $\mathbf{G}'$, and d_n as the n'th diagonal element of $\mathbf{D}$):

$$F(\mathbf{m}_n) = \mathbf{m}'_n \cdot \mathbf{g}'_n - \frac{1}{2}\mathbf{m}'^T_n(\mathbf{Q}_1 + d_n\mathbf{Q}_2)\mathbf{m}'_n, \quad (309)$$

so the solution is:

$$\begin{aligned}\mathbf{m}'_n &= \mathbf{g}'_n(\mathbf{Q}_1 + d_n\mathbf{Q}_2)^{-1} \quad (310)\\ \mathbf{M} &= \mathbf{T}^T\mathbf{M}'. \quad (311)\end{aligned}$$

K. RENORMALIZING THE PHONETIC SUBSPACE

It can be useful to renormalize the phonetic subspace (the space in which the vectors $\mathbf{v}_{jkm}$ lie) on each iteration. This is firstly to avoid numerical issues that can arise if the vectors are too highly correlated between dimensions or have a too large or too small dynamic range, and secondly to concentrate most of the important variation in the lower-numbered dimensions, which is convenient if we want to display the phonetic subspace. If we ignore numerical issues and questions of flooring, condition-limiting and so on that arise secondary to numerical issues, this renormalization makes absolutely no difference at all to the model.

The aim in this renormalization is to ensure that the vectors $\mathbf{v}_{jkm}$ have unit variance and that the most important variation in the vectors is localized to the first dimensions. We ensure this by diagonalizing the matrix $\mathbf{H}_k^{(\mathrm{sm})}$ of Equation (87) and sorting its projected dimensions from largest to smallest. This is appropriate because $\mathbf{H}_k^{(\mathrm{sm})}$ is analogous to a precision matrix (an inverse covariance) and $\mathbf{v}_{jkm}$ is analogous to a mean, so if we have ensured that the means have unit variance if we put the largest elements of the diagonalized precision first it corresponds to having the dimensions with smallest within-class variance first, once the between-class variance is made equal to unity. Some of these equations will look a little different from the typical LDA formulation because we are dealing with an inverse variance like quantity rather than with a variance quantity. We avoid inverting $\mathbf{H}_k^{(\mathrm{sm})}$ because it will be singular before the second iteration of training.

The renormalization is done separately for each sub-model k. For each sub-model k we compute the variance

$$\mathbf{S}_k = \frac{1}{\sum_j \mathbf{M}_{kj}}\sum_j\sum_{m=1}^{\mathbf{M}_j}\mathbf{v}_{jkm}\mathbf{v}_{jkm}^T, \quad (312)$$

which is the variance of the vectors prior to normalization. If $\mathbf{S}_k$ is singular (e.g. its condition is more than 10^{10}) we should skip the renormalization because the vectors are linearly dependent; it probably means we have not yet done any iterations of update. We then do the Cholesky decomposition

$$\mathbf{S}_k = \mathbf{L}\mathbf{L}^T. \quad (313)$$

A transformation that diagonalizes $\mathbf{S}_k$ is now $\mathbf{L}^{-1}$. We then compute the "transformed" $\mathbf{H}_k^{(\mathrm{sm})}$ as:

$$\mathbf{P} = \mathbf{L}^T\mathbf{H}_k^{(\mathrm{sm})}\mathbf{L}. \quad (314)$$

(Note that because $\mathbf{H}_k^{(\mathrm{sm})}$ is a precision-like quantity we must transform with the inverse transpose of the transformation that would apply to a variance-like quantity). We then do the singular value decomposition:

$$\mathbf{P} = \mathbf{U}\mathbf{L}\mathbf{V}^T, \quad (315)$$

which implies that $\mathbf{P} = \mathbf{U}\mathbf{L}\mathbf{U}^T$ because $\mathbf{P}$ is positive semi-definite. We must make sure to sort the diagonal of $\mathbf{L}$ and the corresponding columns of $\mathbf{U}$ from largest to smallest singular value. Now we can clearly diagonalize $\mathbf{H}_k^{(\mathrm{sm})}$ with $\mathbf{U}^T\mathbf{L}^T$, and the appropriate transformation on the vectors themselves is the inverse tranpose of this, which is $\mathbf{U}^T\mathbf{L}^{-1}$ (we use here the fact that $\mathbf{U}^{-1} = \mathbf{U}^T$ because $\mathbf{U}$ is orthogonal). So the transform $\mathbf{F}$ and the resulting updates to the

parameters are below:

$$\begin{aligned}
\mathbf{F} &= \mathbf{U}^T\mathbf{L}^{-1} && (316)\\
\hat{\mathbf{v}}_{jkm} &= \mathbf{F}\mathbf{v}_{jkm} && (317)\\
\hat{\mathbf{w}}_{ki} &= \mathbf{F}^{-T}\mathbf{w}_{ki} && (318)\\
\hat{\mathbf{M}}_{ki} &= \mathbf{M}_{ki}\mathbf{F}^{-1}. && (319)
\end{aligned}$$

It is easy to show that the products $\mathbf{M}_{ki}\mathbf{v}_{jkm}$ and $\mathbf{w}_{ki} \cdot \mathbf{v}_{jkm}$ will be the same before and after this transformation. Note that if we are not using offsets on the vectors $\mathbf{v}_{jkm}$, i.e. we do not have expressions like $\mathbf{v}^+_{jkm}$ then the vectors themselves contain the unit offset term; typically after the transformation described above this unit offset term will to the first dimension in the transformed space so the most significant dimensions for display and visualization purposes would be dimensions two and three.