SUBSPACE GAUSSIAN MIXTURE MODELS FOR SPEECH RECOGNITION

Daniel Povey [*]

IBM T.J. Watson Research Center
Yorktown Heights, NY, USA
{dpovey}@us.ibm.com

ABSTRACT

We introduce speech recognition based on subspace adaptation of a shared Gaussian Mixture Model (GMM). This refers to adaptation to a particular speech state; it is not a speaker adaptation technique, although we do later introduce a speaker adaptation technique that it tied to this particular framework. Our model is a large shared GMM whose parameters vary in a subspace of relatively low dimension (e.g. 50), thus each state is described by a vector of low dimension which controls the GMM's means and mixture weights in a manner determined by globally shared parameters. In addition we generalize to having each speech state be a mixture of substates, each with a different vector. We have previously described speech recognition based on Maximum A Posteriori (MAP) adaptation of a shared Universal Background Model (UBM). We now show that we can get similar gains with subspace adaptation of a GMM (this referred to in the speaker recognition community as a factor analysis approach). The small size of the resulting models makes it possible to do discriminative training, and we show that we can get improvements even on top of a baseline with discriminative training.

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

1. INTRODUCTION

We have previously [1] applied the Universal Background Model (UBM) to speech recognition. In that paper, a shared mixture of diagonal Gaussians was adapted via a tree-based form of MAP (Maximum a Posteriori) estimation, to each speech state. We demonstrated substantial improvements in an ML trained system (i.e. without discriminative training). However, due to the very large number of parameters in the UBM based system we anticipated difficulties training those models discriminatively. Therefore, in this paper we introduce a very different UBM based approach that has fewer parameters, and we show that it can be discriminatively trained and still provide a performance improvement under ML training similar to our previous UBM based approach. What we are introducing here is a subspace approach, in which a vector of low dimension (e.g. 50) controls all the mean and weight parameters of the speech-state-specific mixture model. We also generalize to have a mixture of substates in each state, i.e. each state's distribution is controlled by a number of these 50-dimensional vectors each with its own mixture weight. In addition we introduce a "speaker vector," which introduces an offset to the means (or equivalently, features) that is dependent on the speaker, and exists in a different subspace of the model.

We note that the difference between the work we describe here and our previous work [1] mirrors a difference that exists within the speaker recognition community, where a Maximum A Posteriori (MAP) based approach [2] coexists with an approach similar to our subspace approach that goes by the name factor analysis [3]. The reason for the name is that in that case there is an additional subspace that models the session variability, and therefore the adapted model varies due to both speaker and session specific factors. In our case we also introduce a method that uses two factors; the factor or interest is the speech-state factor, and the factor to be normalized out is the (speaker plus session) factor.

HERE - section specific intro.

2. SUBSPACE MIXTURE MODEL

2.1. Basic model

In this section we describe the Subspace Mixture Model. First we describe the basic model without substates. We use the index $1 \leq i \leq I$ to represent the Gaussians in the UBM (e.g. $I = 750$ Gaussians), and the index $1 \leq j \leq J$ to represent the clustered phonetic states (e.g. $J = 8000$ for a 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. $S = 50$. The subspace dimension can take any value; it represents the number of different directions in which we allow the phonetic states to differ from each other.

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

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

$$\mu_{ji} = \mathbf{M}_i \mathbf{v}_j^+ \tag{2}$$

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

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 denote by $\mathbf{v}_j^+$ 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 $\mathbf{M}_i$, the weight-projection vectors $\mathbf{w}_i$, the variances $\mathbf{\Sigma}_i$, and the state-specific vectors $\mathbf{v}_j$. 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, in reverse order of size: mean-projections, $IDS = 750 \times 40 \times (50 + 1) = 1.53 \times 10^6$; variances, $\frac{1}{2}ID(D+1) = \frac{750 \times 40 \times 41}{2} = 0.615 \times 10^6$; state-specific vectors, $JD = 0.4 \times 10^6$, 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 shared, not state-specific. For reference, a typical mixture-of-Gaussians system might have 100000 Gaussians

[*]This work was funded by DARPA contract HR0011-06-2-0001

in total, each with a 40-dimensional mean and variance, which gives us 8×10^6 parameters total, more than twice this 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 substates.

2.2. Subspace mixture model with substates

The subspace mixture model with substates is the same as in Equations 1 to 3 except each state is now like a mixture of states; each state j has substates $1 \leq m \leq M_j$ 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:

$$p(\mathbf{x}|j) = \sum_{m=1}^{M_j} c_{jm} \sum_{i=1}^{I} w_{jmi} \mathcal{N}(\mathbf{x}; \mu_{jmi}, \mathbf{\Sigma}_i) \quad (4)$$

$$\mu_{jmi} = \mathbf{M}_i \mathbf{v}_{jm}^+ \quad (5)$$

$$w_{jmi} = \frac{\exp \mathbf{w}_i^T \mathbf{v}_{jm}^+}{\sum_{i'=1}^{I} \exp \mathbf{w}_{i'}^T \mathbf{v}_{jm}^+} \quad (6)$$

It is useful to think about the substates as corresponding to Gaussians in a mixture of Gassians, and in fact as we describe later, we use a similar mixing up procedure to increase the number of states. In fact 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 lead to efficiency problems. In fact, computing each mean would involve a matrix multiply taking time $O(SD)$, and since the variances $\mathbf{\Sigma}_i$ are not diagonal the actual likelihood computation would be $O(D^2)$. In the next section we show that despite this, likelihoods given this model can be computed in a time similar to a normal diagonal mixture of Gaussians.

2.3. Subspace mixture model with speaker vectors

Another useful extension to the basic subspace GMM framework is a technique that introduces speaker vectors, where each speaker s will described by a speaker vector v_s of dimension T (in experiments here we use $T = 50$, the same as the subspace dimension S). The speaker subspace of dimension T is analogous to the previously introduced speech-state subspace of dimension S. The projected mean now becomes:

$$\mu_{jmi}^{(s)} = \mathbf{M}_i \mathbf{v}_{jm}^+ + \mathbf{N}_i \mathbf{v}_s^+, \quad (7)$$

so $\mathbf{N}_i \mathbf{v}_s^+$ becomes a speaker-specific offset to the mean. 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 the speech state and the speaker is analogous to the "factor analysis" approach used in speaker identification [3] of having separate subspaces for the speaker and the channel. Because the number of parameters to be estimated per speaker is so small, in practice we actually estimate these vectors for each utterance.

3. FAST EVALUATION OF SUBSPACE MODELS

3.1. Pruning using the UBM

The Universal Background Model (UBM) is a global mixture of Gaussians that is used to initialize the subspace mixture model; think of it as an "unadapted" version of the mixture model. Let the UBM have means $\bar{\mu}_i$ and (full) variances $\bar{\mathbf{\Sigma}}_i$. We use equal mixture weights. Note that the variances $\bar{\mathbf{\Sigma}}_i$ are different from the shared variances $\mathbf{\Sigma}_i$ in the model. We can think of $\bar{\mathbf{\Sigma}}_i$ as a "total variance" and $\mathbf{\Sigma}_i$ as a "within-class" variance; a similar distinction exists where UBMs are used for speaker identification, e.g., see [3]. We prune using the UBM on each frame as follows. First we evaluate all of the UBM Gaussians using only the diagonal of the variance. Then we take e.g. 20 of the most likely Gaussians and evaluate the likelihood using the full covariance matrix. Of these we take, say the top $P = 5$ most likely Gaussians and only perform the summations over i, using the top P indices. This is done during both training and test, and we found that it is best to train and test using the same value of P.

3.2. Fast computation of Gaussian likelihoods

The other issue we have, apart from the very large number of Gaussians, is the fact that the naive Gaussian liklelihood computation without caching Gaussian means (which is impossible due to memory constraints), would take time $O(D^2 + SD)$ for each Gaussian. We show below that we can reduce this to $O(S)$ by doing appropriate precomputations on each frame. Note that typically S and D are similar so this is about twice as fast as evaluating a diagonal Gaussian likelihood which is $O(2D)$. However we are still evaluating more Gaussians in each state in our subspace system than in our baseline: the number is PM_j, rather than M_j in a mixture-of-Gaussians system, and the subspace system might have about half the M_j of our baseline, so the number of floating point operations is about $\frac{1}{2}P$ times our baseline. We find that the total compute time for decoding using subspace models is about two or three times slower than a standard system, given our current setup (the difference might be less if we were not using Gaussian clustering to compute only a subset of Gaussians in each state in our baseline system).

Using $p(\mathbf{x}; j, m, i)$ to denote the contribution to the likelihood of $\mathbf{x}$ given state j from UBM index i and mixture index m, we have:

$$p(\mathbf{x}; j, m, i) = c_{jm} w_{jmi} \exp \Big(-0.5 \left(\log|\det \mathbf{\Sigma}_i| + D\log(2\pi) + (\mu_{jmi} - \mathbf{x})^T \mathbf{\Sigma}_i^{-1} (\mu_{jmi} - \mathbf{x}) \right) \Big) . (8)$$

We can decompose this into a mixture-specific normalizer n_{jmi}, a feature and UBM index-specific normalizer $n_i(\mathbf{x})$ and a cross term:

$$\begin{aligned}
\log p(\mathbf{x}; j, m, i) &= n_{jmi} + n_i(\mathbf{x}) + \mathbf{x}^T \mathbf{\Sigma}_i^{-1} \mu_{jmi} \\
&= n_{jmi} + n_i(\mathbf{x}) + \mathbf{x}^T \mathbf{\Sigma}_i^{-1} \mathbf{M}_i v_{jm} \\
&= n_{jmi} + n_i(\mathbf{x}) + (\mathbf{M}_i^T \mathbf{\Sigma}_i^{-1} \mathbf{x})^T v_{jm} && (9) \\
n_{jmi} &= \log c_{jm} + \log w_{jmi} - 0.5(\log|\det \mathbf{\Sigma}_i| \\
&\quad + D\log(2\pi) + \mu_{jmi}^T \mathbf{\Sigma}_i^{-1} \mu_{jmi}) && (10) \\
n_i(\mathbf{x}) &= -0.5 \mathbf{x}^T \mathbf{\Sigma}_i^{-1} \mathbf{x} && (11) \\
& && (12)
\end{aligned}$$

Thus, the precomputation we must do on every frame involves computing the normalizers $n_i(\mathbf{x})$ and the vectors $\mathbf{M}_i^T \mathbf{\Sigma}_i^{-1} \mathbf{x}$, which can be done in time PD^2 and PSD respectively, which is acceptable. The main cost of this approach is the need to store in memory the normalizers n_{jmi}, which will typically be several times the size of the actual parameters of the model, e.g. for our example system if it had 30000 total mixtures and using floats, it would take 90 MB to store, versus 15.5 MB for the model.

If we are using speaker vectors, this form of adaptation can be implemented efficiently in the above scheme by replacing all instances of $\mathbf{x}$ in Equations 9 to 11 with $(\mathbf{x} - \mathbf{N}_i \mathbf{v}_s^+)$.

4. SUBSPACE MODEL TRAINING

The subspace model training proceeds as follows. Firstly we initialize the UBM, which is a mixture of full-covariance Gaussians that models all speech data regardless of speech state or speaker. Next, we do a first pass of accumulation and update, using a previous system to align speech states to frames. In this first pass of accumulation and update, we are essentially estimating the basic subspace mixture model of Section 2.1, without substates or speaker vectors. In later passes over the data, we accumulate different kinds of statistics and the update equations have a different form.

4.1. UBM initialization

The method we use for initialization of the UBM parameters $\bar{\mu}_i$ and $\bar{\mathbf{\Sigma}}_i$ may not be optimal as we have not experimented with this. We take an already-trained conventional diagonal Gaussian system and cluster the Gaussians into I clusters (e.g. 750). This is done by considering all the Gaussians as one large mixture model (using as weights the weights within each state, divided by the total number of states), and then computing the mixture of I Gaussians that maximizes the auxiliary function likelihood. The algorithm we use to compute this is like a form of k-means except with pruning to avoid excessive compute (this involves a notion of neighboring clusters), starting from a random assignment to clusters. The variances are thus initialized to diagonal. From that point we do 3 iterations of E-M over a subset (e.g. 1/10) of the training data, updating the means and (full) variances but leaving the mixture weights uniform to encourage even distribution of data.

4.2. First pass of training: accumulation

The first pass of training involves getting mean statistics for each state j and UBM index i, and using this to initialize the parameters with a single vector per state. By storing statistics in a different form for the first iteration of update than for later iterations, we can avoid making unnecessary passes over the data. However, to store the mean statistics requires a lot of memory and storage: e.g. for our example system using floats, it would take $4IJD = 4 \times 750 \times 8000 \times 40$ bytes of memory, or 0.96 GB. To reduce this, we avoid storing statistics with very small counts, as we describe below. Our state posteriors $\gamma_j(t)$ are zero-one posteriors based on Viterbi alignments obtained using a baseline (mixture-of-Gaussians) system. On each frame we also compute UBM Gaussian posteriors $\gamma_i(t)$ (with pruning to the top 5 as described above). We then compute initial posteriors:

$$\gamma_{ji}(t) = \gamma_j(t)\gamma_i(t). \tag{13}$$

The statistics we accumulate are count statistics (sums of the posteriors) and state-specific mean statistics, and also a scatter for each UBM Gaussian index which we will use to compute within-class covariances $\mathbf{\Sigma}_i$. There is a slight complication in that we want to avoid accumulating mean statistics where the count is very small. Therefore we define the "pruned" count $\tilde{\gamma}_{ji}(t)$ to be zero if the sum of $\gamma_{ji}(t)$ up to the current point in the current parallel job is less than a threshold τ (we have used τ values from 0.1 to 2 depending on system size). The statistics we accumulate are named $\tilde{\mathbf{m}}_{ji}$ for the first order statistics and $\tilde{S}_i$ for the scatter to emphasize that they are accumulated using the pruned counts. So we have:

$$\begin{aligned}
\gamma_{ji} &= \sum_{t=1}^{T} \gamma_{ji}(t) &(14)\\
\tilde{\gamma}_{ji} &= \sum_{t=1}^{T} \tilde{\gamma}_{ji}(t) &(15)\\
\tilde{\mathbf{m}}_{ji} &= \sum_{t=1}^{T} \tilde{\gamma}_{ji}(t)\mathbf{x}(t) &(16)\\
\tilde{\mathbf{S}}_i &= \sum_{t=1}^{T}\sum_{j=1}^{J} \tilde{\gamma}_{ji}(t)\mathbf{x}(t)\mathbf{x}(t)^T &(17)
\end{aligned}$$

4.3. First pass of training: update

The first pass of update is an iterative one in which we first initialize the vectors to random values (e.g. Gaussian noise), initialize the projections to zero and the variances to the UBM variances, then iteratively optimize in turn each of the four types of parameters: the weight-projection vectors $\mathbf{w}_i$, the mean-projection matrices $\mathbf{M}_i$, the variances $\mathbf{\Sigma}_i$ and the state-specific vectors $\mathbf{v}_j$ (at this point we have no substates). This is done for about ten iterations.

4.3.1. Weight-projection vector update

The update of the weight-projection vectors $\mathbf{w}_i$ is based on maximizing the auxiliary function:

$$\begin{aligned}
\mathcal{Q}(\ldots) &= \sum_i \sum_j \gamma_{ji} \log w_{ji} &(18)\\
&= \sum_{i,j} \gamma_{ji} \left(\mathbf{w}_i^T \mathbf{v}_j - \log \sum_{i'=1}^{I} \exp \mathbf{w}_{i'}{}^T \mathbf{v}_j \right) &(19)
\end{aligned}$$

We can use the inequality $1 - (x/\bar{x}) \leq -\log(x/\bar{x})$ (which is an equality at $x = \bar{x}$), to maximize instead the following auxiliary function, where $\bar{\mathbf{w}}_i$ is the pre-update value of $\mathbf{w}_i$:

$$\mathcal{Q}'(\ldots) = \sum_i \sum_j \gamma_{ji} \left(\mathbf{w}_i^T \mathbf{v}_j - \frac{\sum_{i'=1}^{I} \exp \mathbf{w}_{i'}{}^T \mathbf{v}_j}{\sum_{i'=1}^{I} \exp \bar{\mathbf{w}}_{i'}^T \mathbf{v}_j} \right). \tag{20}$$

To maximize the above we use a second order approximation to the exponential function, but then in certain cases we take a heuristic overestimate of the negated second gradient, for safety; this leads to the $\max(\cdot)$ function below (without this heuristic we would just have its first term). The update procedure is as follows. First we compute all the un-normalized log weights, let us call them $x_{ji} = \mathbf{w}_i^T \mathbf{v}_j$, and the normalizers $x_j = \log \sum_i \exp x_{ji}$; these are used to compute the weights $w_{ji} = \exp(x_{ji} - x_j)$ during the computation. We also compute the total counts per state $\gamma_j = \sum_i \gamma_{ji}$. Then for each UBM Gaussian index i we compute the first order term $\mathbf{g}_i$ and negated second order term $\mathbf{H}_i$ in a quadratic approximation to the auxiliary function in $\mathbf{w}_i - \bar{w}_i$, i.e. around the current point. These are computed as:

$$\begin{aligned}
\mathbf{g}_i &= \sum_j (\gamma_{ji} - \gamma_j w_{ji}) \mathbf{v}_j^+ &(21)\\
\mathbf{H}_i &= \sum_j \max(\gamma_j w_{ji}, \gamma_{ji}) \mathbf{v}_j^+ \mathbf{v}_j^{+T} &(22)\\
\mathbf{w}_i &:= \mathbf{w}_i + \mathbf{H}_i^{-1} \mathbf{g}_i &(23)
\end{aligned}$$

After updating each $\mathbf{w}_i$, we update the affected x_{ji} and the x_j before updating the next i so we can continue with up to date values of w_{ji}. The value of the auxiliary function should be checked as we cannot prove that this procedure will converge, although we have never observed it not converging. In case of nonconvergence we can continue with a slowed-down version of Equation 23.

4.3.2. *Mean-projection matrix update*

The update for the mean-projection matrices $\mathbf{M}_i$ (which have size $D \times S+1$) is as follows. For a particular i, we first make a co-ordinate change so that the variance $\mathbf{\Sigma}_i$ is unit. We use the transform $\mathbf{T} = \mathbf{\Sigma}_i^{-0.5}$, and project to get $\mathbf{M}'_i = \mathbf{T}\mathbf{M}_i$ in the new co-ordinates. Then the computation is as follows: for each of its D rows $\mathbf{m}'_{id}$ we will compute a linear term $\mathbf{g}_{id}$ of the auxiliary function as a function of the change in that row, and a negated quadratic term $\mathbf{H}_i$ which is shared for all d.

$$\mathbf{g}_{id} = \sum_j \left(\mathbf{T}\tilde{\mathbf{m}}_{ji} - \tilde{\gamma}_{ij}\mathbf{M}'_i\mathbf{v}_j^+\right)_d \mathbf{v}_j^+ \quad (24)$$

$$\mathbf{H}_i = \sum_j \tilde{\gamma}_{ji}\mathbf{v}_j^+\mathbf{v}_j^{+T} \quad (25)$$

$$\mathbf{m}'_{id} := \mathbf{m}'_{id} + \mathbf{H}_i^{-1}\mathbf{g}_d. \quad (26)$$

The auxiliary function improvement is $0.5\mathbf{g}_d^T\mathbf{H}_i^{-1}\mathbf{g}_d$. We project back to get $\mathbf{M}_i := \mathbf{T}^{-1}\mathbf{M}'_i$.

4.3.3. *Variance updates*

The update for the variances $\mathbf{\Sigma}_i$ is very simple:

$$\mathbf{\Sigma}_i := \frac{\tilde{\mathbf{S}}_i + \sum_{j=1}^J -\tilde{\mathbf{m}}_{ji}\mu_{ji}^T - \mu_{ji}\tilde{\mathbf{m}}_{ji}^T + \tilde{\gamma}_{ji}\mu_{ji}\mu_{ji}^T}{\sum_{j=1}^J \tilde{\gamma}_{ji}}, \quad (27)$$

where $\mu_{ji} = (\mathbf{M}_i\mathbf{v}_j^+)$. If we use $\hat{\mathbf{\Sigma}}_i$ to represent the post-update variance and $\mathbf{\Sigma}_i$ is the pre-update variance, the auxiliary function improvement is given by $-0.5\left(\sum_{j=1}^J \tilde{\gamma}_{ji}\right)\left(\log|\det\hat{\mathbf{\Sigma}}_i| - \log|\det\mathbf{\Sigma}_i| + tr((\hat{\mathbf{\Sigma}}_i^{-1} - \mathbf{\Sigma}_i^{-1})\hat{\mathbf{\Sigma}}_i)\right)$. We recommend inspecting all auxiliary function improvements for diagonstic purposes and to check for convergence.

4.3.4. *Vector updates*

The update for the state-specific vectors $\mathbf{v}_j$ involves incorporating a quadratic auxiliary function for the means, and our previously described quadratic approximation to the auxiliary function for the weights. Again we accumulate a linear term $\mathbf{g}_j$ and a negated quadratic term $\mathbf{H}_j$ which describe how the auxiliary function varies with a *change* in $\mathbf{v}_j$. In the expressions below, the top line in each expression refers to the weights and the bottom line to the means. We use the notation $\mathbf{x}^-$ to mean the vector $\mathbf{x}$ without its last element; for matrices the notation $\mathbf{M}^-$ means removing the last row and column.

$$\begin{aligned}\mathbf{g}_j &= \sum_{i=1}^I(\gamma_{ji} - \gamma_j w_{ji})\mathbf{w}_i^- \\ &\quad + \sum_{i=1}^I\left(\mathbf{M}_i^T\mathbf{\Sigma}_i^{-1}(\tilde{m}_{ji} - \tilde{\gamma}_{ji}\mathbf{M}_i\mathbf{v}_j^+)\right)^- \quad (28)\\ \mathbf{H}_j &= \sum_{i=1}\max(\gamma_{ji}, \gamma_j w_{ji})\mathbf{w}_i^-\mathbf{w}_i^{-T} \\ &\quad + \sum_{i=1}^I \gamma_{ji}\left(\mathbf{M}_i^T\mathbf{\Sigma}_i^{-1}\mathbf{M}_i\right)^- \quad (29)\\ \mathbf{v}_j &:= \mathbf{v}_j + \mathbf{H}_j^{-1}\mathbf{g}_j \quad (30)\end{aligned}$$

The matrices only dependent on i in the last line of Equation 29 should be precomputed.

4.4. Later iterations of training: accumulation

The method of accumulation differs in later iterations of training, versus the first iteration. We store statistics in a more memory efficient way, without pruning. This enables a more exact optimization, and also allows us to have more mixtures without increasing the size of the statistics too much. The size of the statistics are dominated by the need to store data counts for each i, j and m. For these later iterations we assume that we already have a "substate" model; we initialize this by having a single substate per state as estimated above, and using unit weight. The state posteriors are, as before, zero-one posteriors based on Viterbi alignment using a previous system.

4.4.1. *Discretized posteriors*

The within-state posteriors $\gamma_{jmi}(t)$ are computed by evaluating the likelihoods as described in Section 3.2. However, we also randomly discretize the posteriors into steps of typically $\delta = 0.05$. This reduces compute time by getting rid of most very small posteriors, and also allows us to compress the posteriors in memory and on disk in a variable length coding scheme in which counts γ_{jmi} typically take only one byte to store. The discretized posteriors $\tilde{\gamma}_{jmi}(t)$ consist of the part of $\gamma_{jmi}(t)$ that can be expressed in whole increments of δ, plus with probability equal to the remaining part divided by δ, one extra increment of δ. The random element of the discretization process is necessary to preserve expectations. All statistics are stored using the discretized posteriors.

4.4.2. *Statistics*

The weight statistics are straightforward:

$$\gamma_{jmi} = \sum_{t=1}^T \tilde{\gamma}_{jmi}(t) \quad (31)$$

The statistics we store in order to update the vectors $\mathbf{v}_{jm}$ are the first order term in the quadratic auxiliary function written in terms of the $\mathbf{v}_{jm}$ directly (i.e. not in terms of offsets from the current value). Again, $\mathbf{x}^-$ is $\mathbf{x}$ without its last dimension. So we have:

$$\mathbf{x}_{jm} = \sum_{t=1}^T\sum_{i=1}^T \tilde{\gamma}_{jmi}(t)\left(\mathbf{M}_i^T\mathbf{\Sigma}_i^{-1}\mathbf{x}(t)\right)^- \quad (32)$$

The statistics we store in order to update the mean projection matrices $\mathbf{M}_i$ are of a similar nature:

$$\mathbf{X}_i = \sum_{t=1}^{T} \sum_{j=1}^{J} \sum_{m=1}^{M_j} \tilde{\gamma}_{jmi}(t) \left(\mathbf{\Sigma}_i^{-1} \mathbf{x}(t) \right) \mathbf{v}_{jm}^T \tag{33}$$

The statistics we store in order to update the variances are simply the variance of the data around the current model means, for each UBM Gaussian index:

$$\mathbf{S}_i = \sum_{t,j,m} \tilde{\gamma}_{jmi}(t) (\mathbf{x}(t) - \mathbf{M}_i \mathbf{v}_{jm})(\mathbf{x}(t) - \mathbf{M}_i \mathbf{v}_{jm})^T \tag{34}$$

4.5. Later iterations of training: update

The update for later iterations of training is somewhat harder to justify than the update for the first iteration. The reason is that there are updates which we do at the same time (for the variance, the vectors and the mean projections) which cannot easily be proved to converge unless they are done on separate iterations. However, we are confident that these parameter types are sufficiently orthogonal that this is not a problem, and in practice we find that our approach converges. Note that when any the updates below refer to other types of parameters (e.g. if the update for $\mathbf{M}_i$ refers to $\mathbf{v}_{jm}$), this means the pre-update versions of those parameters. This is important because the stored statistics are a function of the other parameters, and using the newly updated versions can lead to inconsistency.

4.5.1. Weight-projection vector update

The update for the weight projection vectors is the same as that described in Section 4.3.1, except that we have to replace any sums over j with sums over both j and m. We do the update for up to 4 iterations given the stored statistics, or until the auxiliary function improvement per frame is small (e.g. less than 0.0001).

4.5.2. Mean-projection matrix update

The update for the mean-projection matrix is similar to that given in Section 4.3.2 except we formulate the quadratic auxiliary function in terms of the transformed matrix row $\mathbf{m}'_{id}$ rather than the offset from its current value. Again we use the data transform $\mathbf{T} = \mathbf{\Sigma}_i^{-0.5}$ to make the variances unit, so $\mathbf{M}'_i = \mathbf{T}\mathbf{M}_i$.

$$\begin{aligned} \mathbf{g}_{id} &= \text{transposed d'th row of } \mathbf{T}\mathbf{X}_i & (35)\\ \mathbf{H}_i &= \sum_j \tilde{\gamma}_{jmi} \mathbf{v}_{jm}^+ \mathbf{v}_{jm}^{+\,T} & (36)\\ \hat{\mathbf{m}'}_{id} &= \mathbf{H}_i^{-1} \mathbf{g}_{id}. & (37) \end{aligned}$$

The auxiliary function improvement for i, d is given by $\mathbf{g}_{id}^T(\hat{\mathbf{m}'}_{id} - \mathbf{m}'_i d) - 0.5 \left(\hat{\mathbf{m}}_{id} \mathbf{H}_i \hat{\mathbf{m}}_{id} - \mathbf{m}_{id}^T \mathbf{H}_i \mathbf{m}_{id} \right)$, where $\mathbf{m}_{id}$ are the pre-update rows. Again, we project back to get $\hat{\mathbf{M}}_i := \mathbf{T}^{-1} \hat{\mathbf{M}}'_i$.

4.5.3. Vector update

In the vector update as follows, we split the second gradient $\mathbf{H}_j$ into two parts that relate to the weights and the means respectively, and use the second one $\mathbf{H}_j^{(2)}$ in our computation of the gradient to convert from a formulation in terms of the vector $\mathbf{v}_{jm}$, to the change in the vector. We make use of the summed counts $\gamma_{jm} = \sum_{i=1}^{I} \gamma_{jmi}$. The update is:

$$\begin{aligned} \mathbf{H}_{jm}^{(1)} &= \sum_{i=1} \max(\gamma_{jmi}, \gamma_{jm} w_{jmi}) \mathbf{w}_i^- \mathbf{w}_i^{-\,T} & (38)\\ \mathbf{H}_{jm}^{(2)} &= \sum_{i=1}^{I} \gamma_{jmi} \left(\mathbf{M}_i^T \mathbf{\Sigma}_i^{-1} \mathbf{M}_i \right)^- & (39)\\ \mathbf{g}_{jm} &= \sum_{i=1}^{I} (\gamma_{jmi} - \gamma_{jm} w_{jmi}) \mathbf{w}_i^- \\ &\quad + \mathbf{x}_{jm} - \mathbf{H}_{jm}^{(2)} \mathbf{v}_{jm} & (40)\\ \mathbf{H}_{jm} &= \mathbf{H}_{jm}^{(1)} + \mathbf{H}_{jm}^{(2)} & (41)\\ \hat{\mathbf{v}}_{jm} &:= \mathbf{v}_{jm} + \mathbf{H}_{jm}^{-1} \mathbf{g}_{jm}. & (42) \end{aligned}$$

Again we precompute the quantity $\mathbf{M}_i^T \mathbf{\Sigma}_i^{-1} \mathbf{M}_i$. The auxiliary function improvement is given by $0.5 \mathbf{g}_{jm}^T \mathbf{H}_{jm}^{-1} \mathbf{g}_{jm}$.

4.5.4. Variance update

The variance update is trivial:

$$\hat{\mathbf{\Sigma}}_i = \frac{\mathbf{S}_i}{\sum_{j,m} \gamma_{jmi}}. \tag{43}$$

The auxiliary function improvement can be computed as described in Section 4.3.3.

4.5.5. Substate weight

We now have a new parameter to estimate: the weight of substates. This is given by:

$$c_{jm} = \frac{\sum_i \gamma_{jmi}}{\sum_{i,m} \gamma_{jmi}} \tag{44}$$

4.5.6. Mixing up

Here we describe how we increase the number of substates. The initial model has one substate per state. We have a target total number of mixtures per state, e.g. $M = 50,000$ and we allocate mixture components to states based on a power rule with a default exponent of 0.2. Thus, if a state has total count $\gamma_j = \sum_{m,i} \gamma_{jmi}$, the target number of mixture components T_j is the closest integer to $M \frac{\gamma_j^{0.2}}{\sum_j \gamma_j^{0.2}}$. We do mixing up on a subset of iterations (currently {2,4,6,8,10,12}). On each iteration and for each state j, the number of mixture components to split shall be the difference between the target T_j and the current number of mixture components M_j; but no more than the current M_j. If it is less than that, we split those with the largest counts. In addition, we enforce a minimum count for mixtures to be split, which is 200 by default. For each substate vector $\mathbf{v}_{jm}$ that is selected to be split, we compute the negated second gradient $\mathbf{H}_{jm}$ as used in section 4.5.3, and then compute the scale $\mathbf{S} = \left(\frac{\mathbf{H}_{jm}}{\gamma_{jm}} \right)^{-0.5}$, which provides a scale to the vector (think of $\mathbf{S}$ like a standard deviation). We then compute a random vector $\mathbf{r}$ whose elements are drawn from zero-mean Gaussian distribution with variance 0.1, and our perturbed vectors shall be $\mathbf{v}_{jm} \pm \mathbf{S}\mathbf{r}$. We assign half of the old mixture weight to each of the two new mixture components. Mixing up is done after all other phases of update are complete (i.e., starting from the already updated vectors).

4.5.7. Updating the UBM

The UBM parameters $\bar{\mu}_i$ and $\bar{\mathbf{\Sigma}}_i$ which are used for pruning are also updated in our training setup. This is done by accumulating zeroth, first and second order statistics for each i and doing the normal Gaussian update. The posteriors used are the sum over substate j, m of the posteriors $\gamma_{jmi}(t)$. Because of the discrete nature of the pruning operation it is not easy to say very much theoretically about how these parameters should be trained, in fact it might seem safer to leave them fixed. Experiments have failed to show any difference between training and not training these parameters.

5. ADAPTATION

5.1. Speaker vectors

The computation of the speaker vector $\mathbf{v}_s$ for a particular speaker (or, in practice, a particular utterance) is done as follows. The statistics consist of the linear term $\mathbf{g}_s$ in the objective function in terms of the change in $\mathbf{v}_s$, plus the count γ_{is} per Gaussian in the UBM (so the statistics per speaker are very small, of dimension $T + I$). The statistics and the update are as follows, where the sum over time is understood to only cover the speaker in question:

$$\gamma_{is} = \sum_{t,j,m} \gamma_{jmi}(t) \tag{45}$$

$$\mathbf{g}_s = \sum_{t,j,m} \left(\mathcal{N}_i^T \Sigma_i^{-1} (\mathbf{x}(t) - \mathbf{N}_i \mathbf{v}_s - \mu_{jmi}) \right)^{-} \tag{46}$$

$$\mathbf{H}_s := \sum_i \gamma_{is} \left(\mathcal{N}_i^T \Sigma_i^{-1} \mathcal{N}_i \right)^{-} \tag{47}$$

$$\mathbf{v}_s := \mathbf{v}_s + \mathbf{g}_s \mathbf{H}_s^{-1} \tag{48}$$

Where $\mathbf{v}_s$ appears above, this is the pre-update value of $\mathbf{v}_s$ which on the first iteration would be zero; we see a small improvement from doing two iterations of the update (using the same word sequence but different posteriors). The matrix term in parentheses in Equation 47 should be precomputed and reused for all speakers.

5.2. MLLR

Currently our subspace mixture model system is trained and tested using VTLN warp factors and fMLLR/CMLLR transforms obtained using our baseline. Thus we avoid having to implement these algorithms within the UBM framework. We have implemented MLLR adaptation in the UBM framework, which sometimes seems to give slightly better results than using a (single) MLLR transform obtained from a baseline system. MLLR also appears to give most of its normal improvement even after using the speaker vectors as described above, which is somewhat surprising as the techniques are quite similar. Since the UBM is in effect a full covariance model, we had to modify our algorithms, and in fact the accumulation phase is now quicker than in the baseline. The statistics we store consist of i) statistics the same size as the MLLR transform, which basically contain the linear term of the objective function as a function of the transform parameters and ii) a set of full-covariance statistics, one for each UBM Gaussian index, which consist of weighted outer products of [extended] mean vectors, including a count for each UBM Gaussian index. The MLLR computation given these statistics is an iterative row-by-row computation. A similar scheme would be possible for fMLLR, in which the full-covariance statistics would be over [extended] data vectors rather than means. The general approach is described in [4], although note that there is an error in Section 3.1 (missing data counts). We use a single regression class as the improvement we get from using multiple classes is very small. Decoding using MLLR is very slow due to the need to recompute the normalizers n_{jmi} and gives very little improvement on top of the speaker factor approach described below, so it is not essential to include. In addition, the WER difference between estimating the MLLR transform using the UBM framework directly versus simply inheriting it from a conventional system is small if it exists at all.

6. DISCRIMINATIVE TRAINING

6.1. Model-space discriminative training: overview and accumulation

The discriminative training experiments reported here are done with boosted MMI [5], although we could also use MPE or normal MMI. Because the final auxiliary functions that we optimize during training are mostly quadratic functions of the parameter, it is possible to formulate an Extended Baum-Welch like update as a very simple rule, which says: take the difference of the linear terms $\mathbf{g}$ of the numerator and denominator auxiliary functions, but the sum of the quadratic terms $\mathbf{H}$. First, we must make sure that the auxiliary function is formulated in terms of the change in parameter, rather than the parameter itself; this ensures that the auxiliary function formulated as described has the correct gradient. In addition we set a speed parameter s for each of the parameter types, e.g. s_w for the weight parameters, etc., and divide the quadratic term in the auxiliary function by this speed.

In discriminative training, since we start from an ML-trained system we only need the non-initial style of accumulation and update. The accumulation is quite a simple modification of the ML accumulation as described in Section 4.4. On each frame we have both denominator and numerator occupation probabilities $\gamma_{jmi}^{\text{num}}(t)$ and $\gamma_{jmi}^{\text{den}}(t)$, obtained (for boosted MMI training) after forward-backward alignment of numerator and denominator lattices and canceling of statistics on each frame as described in [5]. The randomized discretization described in Section 4.4.1 is applied to both numerator and denominator occupation probabilities. There is no I-smoothing or equivalent, so we have no ML statistics. We duplicate our count statistics into numerator and denominator counts, but store the other statistics in combined form:

$$\gamma_{jmi}^{\text{num}} = \sum_{t=1}^{T} \tilde{\gamma}_{jmi}^{\text{num}}(t) \tag{49}$$

$$\gamma_{jmi}^{\text{den}} = \sum_{t=1}^{T} \tilde{\gamma}_{jmi}^{\text{den}}(t) \tag{50}$$

$$\mathbf{x}_{jm} = \sum_{t=1}^{T} \sum_{i=1}^{T} (\tilde{\gamma}_{jmi}^{\text{num}}(t) - \tilde{\gamma}_{jmi}^{\text{den}}(t)) \left(\mathbf{M}_i^T \mathbf{\Sigma}_i^{-1} \mathbf{x}(t) \right)^{-} \tag{51}$$

$$\mathbf{X}_i = \sum_{t,j,m} (\tilde{\gamma}_{jmi}^{\text{num}}(t) - \tilde{\gamma}_{jmi}^{\text{den}}(t)) \left(\mathbf{\Sigma}_i^{-1} \mathbf{x}(t) \right) \mathbf{v}_{jm}^T \tag{52}$$

$$\mathbf{S}_i = \sum_{t,j,m} (\tilde{\gamma}_{jmi}^{\text{num}}(t) - \tilde{\gamma}_{jmi}^{\text{den}}(t)) (\mathbf{x}(t) - \mathbf{M}_i \mathbf{v}_{jm})(\mathbf{x}(t) - \mathbf{M}_i \mathbf{v}_{jm})^T. \tag{53}$$

6.2. Model-space discriminative training: update

As with the Maximum Likelihood update described above, it is important that the update equations for each parameter type (weight-projections, mean-projections, etc.) "see" only the pre-update values of the other parameter types. We have avoided making this explicit with iteration indices as this would clutter the equations.

6.2.1. *Weight-parameter update*

The discriminative weight parameter update is as follows, controlled by speed parameter s_w, e.g. $s_w = 1.0$. We use the total per-mixture counts e.g. $\gamma_{jm}^{\rm num} = \sum_{i=1}^{I} \gamma_{jmi}^{\rm num}$.

$$
\begin{aligned}
\mathbf{g}_i &= \sum_{j,m} ((\gamma_{jmi}^{\rm num} - \gamma_{jmi}^{\rm den}) - (\gamma_{jm}^{\rm num} - \gamma_{jm}^{\rm den}) w_{jmi}) \mathbf{v}_j^{+} && (54) \\
\mathbf{H}_i &= \textstyle\sum_{j,m} \left(\max(\gamma_{jm}^{\rm num} w_{jmi}, \gamma_{jmi}^{\rm num}) + \max(\gamma_{jm}^{\rm den} w_{jmi}, \gamma_{jmi}^{\rm den}) \right) \\
& \qquad \mathbf{v}_j^{+} \mathbf{v}_j^{+T} && (55) \\
\mathbf{w}_i &:= \mathbf{w}_i + s_w \mathbf{H}_i^{-1} \mathbf{g}_i. && (56)
\end{aligned}
$$

The auxiliary function improvement is $0.5 s_w \mathbf{g}_i \mathbf{H}_i^{-1} \mathbf{g}_i$[1]. Note that the values of w_{jmi} we use in this discriminative update are always the pre-update values. This update it not iterative, being applied only once during each update phase.

6.2.2. *Mean-projection matrix update*

For the mean-projection matrix update, again we use the normalizing transform $\mathbf{T} = \mathbf{\Sigma}_i^{-0.5}$, and project to get $\mathbf{M}'_i = \mathbf{T}\mathbf{M}_i$.

$$
\begin{aligned}
\mathbf{g}_{id}^{\rm abs} &= \text{transposed d'th row of } \mathbf{T}\mathbf{X}_i && (57) \\
\mathbf{H}_i^{\rm num} &= \sum_j \tilde{\gamma}_{jmi}^{\rm num} \mathbf{v}_{jm}^{+} \mathbf{v}_{jm}^{+\ T} && (58) \\
\mathbf{H}_i^{\rm den} &= \sum_j \tilde{\gamma}_{jmi}^{\rm num} \mathbf{v}_{jm}^{+} \mathbf{v}_{jm}^{+\ T} && (59) \\
\mathbf{g}_{id} &= \mathbf{g}_{id}^{\rm abs} - (\mathbf{H}_i^{\rm num} - \mathbf{H}_i^{\rm den}) \mathbf{m}'_{id} && (60) \\
\mathbf{H}_i &= \left(\mathbf{H}_i^{\rm num} + \mathbf{H}_i^{\rm den} \right) && (61) \\
\hat{\mathbf{m}'}_{id} &:= \mathbf{m}'_{id} + s_m \mathbf{H}_i^{-1} \mathbf{g}_{id}. && (62)
\end{aligned}
$$

The auxiliary function improvement is given by $0.5 s_m \mathbf{g}_{id}^T \mathbf{H}_i^{-1} \mathbf{g}_{id}$. Again we transform back by setting $\mathbf{M}_i := \mathbf{T}^{-1} \mathbf{M}'_i$.

[1]This formula assumes that we apply s_w at the auxiliary function level, to the second gradient $\mathbf{H}_i$.

6.2.3. *Vector update*

In the discriminative vector update we precompute summed counts such as $\gamma_{jm}^{\rm num} = \sum_{i=1}^{I} \gamma_{jmi}^{\rm num}$. The update is:

$$
\begin{aligned}
\mathbf{H}_{jm}^{(1)} &= \sum_{i=1} \max(\gamma_{jmi}^{\rm num}, \gamma_{jm}^{\rm num} w_{jmi}) \mathbf{w}_i^{-} \mathbf{w}_i^{-T} \\
& \quad + \max(\gamma_{jmi}^{\rm den}, \gamma_{jm}^{\rm den} w_{jmi}) \mathbf{w}_i^{-} \mathbf{w}_i^{-T} && (63) \\
\mathbf{H}_{jm}^{(2,\rm num)} &= \sum_{i=1}^{I} \gamma_{jmi}^{\rm num} \left(\mathbf{M}_i^T \mathbf{\Sigma}_i^{-1} \mathbf{M}_i \right)^{-} && (64) \\
\mathbf{H}_{jm}^{(2,\rm den)} &= \sum_{i=1}^{I} \gamma_{jmi}^{\rm den} \left(\mathbf{M}_i^T \mathbf{\Sigma}_i^{-1} \mathbf{M}_i \right)^{-} && (65) \\
\mathbf{g}_{jm} &= \sum_{i=1}^{I} (\gamma_{jmi} - \gamma_{jm} w_{jmi}) \mathbf{w}_i^{-} \\
& \quad + \mathbf{x}_{jm} - (\mathbf{H}_{jm}^{(2)\rm num} - \mathbf{H}_{jm}^{(2)\rm den}) \mathbf{v}_{jm} && (66) \\
\mathbf{H}_{jm} &= \frac{1}{s_v} \left(\mathbf{H}_{jm}^{(1)} + \mathbf{H}_{jm}^{(2)\rm num} + \mathbf{H}_{jm}^{(2)\rm den} \right) && (67) \\
\hat{\mathbf{v}}_{jm} &:= \mathbf{v}_{jm} + \mathbf{H}_{jm}^{-1} \mathbf{g}_{jm}. && (68)
\end{aligned}
$$

Again we cache the quantity $\mathbf{M}_i^T \mathbf{\Sigma}_i^{-1} \mathbf{M}_i$. The auxiliary function improvement is given by $0.5 \mathbf{g}_{jm}^T \mathbf{H}_{jm}^{-1} \mathbf{g}_{jm}$.

6.2.4. *Variance update*

For the discriminative variance update, let $\gamma_i^{\rm num} = \sum_{j,m} \gamma_{jmi}^{\rm num}$ and similarly for the denominator counts. We set $E = 2.0/s_\sigma$, to translate from the "speed" notation to the notation of Extended Baum-Welch updates. For each i, we set $D_i = E\gamma_i^{\rm den}$, and the update is:

$$
\hat{\mathbf{\Sigma}}_i = \frac{\mathbf{S}_i + D_i \mathbf{\Sigma}}{\gamma_i^{\rm num} - \gamma_i^{\rm den} + D_i}. \qquad (69)
$$

As an additional check, we make sure that the updated $\hat{\mathbf{\Sigma}}_i$ is positive definite when computed as above but with D_i at half its value. If not, we increase D_i in increments until this condition holds. The total auxiliary function improvement is: $\sum_i -0.5 \left(\gamma_i^{\rm num} - \gamma_i^{\rm den} + E \right)$ $\left(\log|\det \hat{\mathbf{\Sigma}}_i| - \log|\det \mathbf{\Sigma}_i| + tr((\hat{\mathbf{\Sigma}}_i^{-1} - \mathbf{\Sigma}_i^{-1}) \hat{\mathbf{\Sigma}}_i) \right)$.

6.2.5. *Substate weight and UBM parameter updates.*

We do not bother updating the substate weights discriminatively, since we anticipate that the effect of this will be small. We also do not update the UBM parameters $\bar{\mu}_i$ and $\bar{\mathbf{\Sigma}}_i$.

6.2.6. *Limiting parameter changes*

The discriminative update as described so far seems to be prone to instability. This is apparent in various ways: we the objective function starting to fall after climbing for a few iterations, and the word error rate starting to degrade. This can be somewhat improved by decreasing most of the speeds s from their default values of 1, e.g. we favor the settings $s_m = 0.5, s_v = 0.5, s_\sigma = 0.5, s_w = 2$. However the word error rate still degrades substantially after reaching its optimim in about four iterations, and if we look at the auxiliary function improvements for different UBM Gaussian indices i we see that for some values of i these improvements rise with iteration number and reach very high values, indicating likely divergence. In order to ameliorate this problem we introduced limits on the maximum auxiliary function improvement per frame, for the four different kinds of parameters. The number of frames in this case is defined as the total numerator plus denominator count, i.e. $\gamma_i^{\rm num} + \gamma_i^{\rm den}$ where $\gamma_i^{\rm num} = \sum_{j,m} \gamma_{jmi}^{\rm num}$, etc. We define a maximum auxiliary function improvement per frame for different parameter classes; these are obtained from viewing a scatter plot the measured auxiliary function improvements per frame for these parameter classes for different i against the counts for the same i, and looking at the maximum of the bulk of the distribution as the counts become reasonably large. These limits are: 0.01 for weight projections, 0.01 for variances, 0.04 for mean projections, and 0.1 for vectors. If for a particular i the measured auxiliary function improvement per frame exceeds this by a certain factor f, we scale the parameter change by $1/f$. In the case of the vectors this statement must be applied for a particular substate j, m. The effect on WER on the best iteration of this limiting operation is minimal, but it does slow the pace of the eventual degradation.

6.2.7. Feature-space discriminative training

Feature-space discriminative training, introduced in [6] but using the recipe from [7] which is significantly better, is an important part of our baseline system, e.g. see [5] for typical improvements on a variety of setups. We find that feature space discriminative training gives less improvement in the subspace mixture model system than with our baseline system; in fact, the gains of feature-space plus model-space versus model-space only training are very marginal. We believe there are two possible reasons for this. One is that the Gaussian mixture model used for fMPE/fMMI is very similar to the Gaussian mixture model used for the UBM (in fact they are obtained in basically the same way), and the two techniques may be exploiting similar kinds of phenomena. Another is that since we are in effect using full covariance models, any linear transformations of the feature space that fMPE/fMMI might be doing will have no effect because the models will simply move to compensate. In Section 7 we do give results with feature space discriminative training but since it appears to be a poor combination we will only describe the outline of our implementation.

Computation of the "direct differential" is very simple. The "indirect differential" involves some choices, and rather than attempt to justify them we will simply state our approach: we compute the indirect differential via only the within-class variances $\mathbf{\Sigma}_i$ and the mean-projection matrices $\mathbf{M}_i$. This can be done starting from the same statistics that we use for normal discriminative training. When updating the model with ML on each iteration, we only update the within-class variances and mean-projection matrices. Every other aspect of the computation is the same as in our baseline fMPE/fMMI computation.

7. EXPERIMENTAL RESULTS

Blah

8. CONCLUSIONS

Blah

9. REFERENCES

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

[2] Reynolds D. A, Quatieri T. F and Dunn R. , "Speaker verification using adapted Gaussian mixture models," *Digital Signal Processing*, vol. 10, no. 1-3, pp. 19–41, 2000.

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

[5] Povey D., Kanevsky D., Kingsbury B., Ramabhadran B., Saon G. and Visweswariah K., "Boosted MMI for Feature and Model Space Discriminative Training," in *ICASSP*, 2008.

[6] Povey D., Kingsbury B., Mangu L., Saon G., Soltau H., and Zweig G., "fMPE: Discriminatively trained features for speech recognition," in *ICASSP*, 2005.

[7] Povey D., "Improvements to fMPE for discriminative training of features," in *Interspeech*, 2005.