Adaptation of front end parameters in a speech recognizer

K. Visweswariah, R. Gopinath

IBM T. J. Watson Research Center, Yorktown Heights, NY - 10598
{kv1, rameshg}@us.ibm.com

Abstract
In this paper we consider adapting the parameters of the algorithm used for extraction of features. Typical speech recognition systems use a sequence of modules to extract features which are then used for recognition. We present a method to adapt the parameters in these modules under a variety of criteria, e.g. maximum likelihood, maximum mutual information. This method works under the assumption that the functions that the modules implement are differentiable with respect to their inputs and parameters. We use this framework to optimize a linear transform preceding the linear discriminant analysis (LDA) matrix and show that it gives significantly better performance than a linear transform after the LDA matrix with small amounts of data. We show that linear transforms can be estimated by directly optimizing likelihood or the MMI objective without using auxiliary functions. We also apply the method to optimize the Mel bins, and the compression power in a system that uses power law compression.

1. Introduction
State of the art speech recognition systems typically adapt the acoustic models and/or the features based on the data seen in test conditions to get improved recognition accuracy. Popular ways of adapting the acoustic model include MAP adaptation [1], MLLR [2]. Methods for adapting the features include linear feature space transforms [3], non-linear transforms [4] and vocal tract length normalization (VTLN) [5].

In this paper we consider adapting the features by adapting any of the parameters in the various feature extraction modules. Feature extraction for speech recognition typically consists of the following modules: magnitude spectrum extraction, Mel binning, a nonlinearity (typically the logarithm), the discrete cosine transform, mean normalization, splicing followed by a projection and finally a linear transform to make the diagonal Gaussian assumption more valid. Most of these modules have parameters that have been set either by training them on the available training data or have been set by experiments on the human auditory system. It is plausible that adapting these modules in a principled manner based on data in test conditions would give us improvements in the performance of a speech recognition system. Since we allow adaptation of all modules of the feature extraction process we subsume techniques such as adaptation with linear transforms, non-linear transforms and in principle even VTLN. We use second order general purpose optimization techniques to perform the optimization. Using these algorithms necessitates the calculation of the objective function and its gradient efficiently. We use the forward-backward algorithm along with the chain rule to calculate all requisite gradients. The methods of this paper are based on [6] where this is applied to building a document recognition system. The implementation we use is object oriented with each front end module performing calculations required to propagate the gradient independent of the other modules. This allows for flexibility in arranging the modules in any order, and adding and removing modules without having to rewrite amounts of code to reimplement the calculation of the gradients.

The rest of the paper is organized as follows. In Section 2 we describe the method for estimating the parameters given a fixed objective function. Section 3 discusses the choice of objective function, and the gradient calculations we need to make for specific choices of objective functions. In Section 4 we explicitly give the gradients for various modules in the front end that we consider. Section 5 describes our experimental setup and results.

2. Overview of the methodology to adapt the front end
The problem we consider in this paper is to update parameters in the front end given some test utterances of raw acoustic data. We may or may not be given the true text of what was said. The supervised case, when we are given the true text, is useful in enrollment type scenarios where the user is asked to read a given script and this data is used to improve the performance of the system for future use by this user.

Let the \( n \)th utterance of this raw acoustic test data be denoted by \( r^{(n)} \). For each utterance \( n \) the final features used by the acoustic model are obtained by transformation by a sequence of \( M \) modules each with its (possibly empty) set of parameters as follows: 

\[
g_{0}^{(n)} = r^{(n)}, \quad g_{1}^{(n)} = f_{i}(g_{i-1}^{(n)}, \theta_{i}), \quad x^{(n)} = g_{M}^{(n)}.
\]

We assume that we have some cost function of the final feature vectors \( x^{(n)} \) that we want to minimize that is of the form:

\[
g(x_{1}^{N}) = \sum_{n=1}^{N} g(x^{(n)}).
\]

This cost function can be a function of the acoustic model and the set of possible word sequences. Typically \( g \) is taken to be the negative log likelihood of the features under the given acoustic model and given a single word sequence. The choice of \( g \) that is appropriate will be discussed in Section 3.

We use the limited memory BFGS algorithm [7] with the More-Thuente line search algorithm [8] as implemented in [9] to minimize \( g \). This requires computation of \( g \) and its gradient with respect to all of the parameters of interest in the front end. Assuming we can calculate gradient \( g \) w.r.t. \( x^{(n)} \) then we can propagate this gradient using the chain rule to calculate all
required gradients as follows:
\[
\frac{dg(x)}{d\theta_i} = \frac{dg(x)}{dy_i} \cdot \frac{dy_i}{d\theta_i}
\]
(1)
and
\[
\frac{dg(x)}{dy_{i-1}} = \frac{dg(x)}{dy_i} \cdot \frac{dy_i}{gy_{i-1}}
\]
(2)

We start at the final module and propagate the gradients through all previous modules. Just as in the feature calculation we start with the raw acoustic data and calculate final feature by using the output of a module as input to the next, for the gradients we do a similar thing in reverse. Clearly we need to assume that all modules that have parameters must implement differentiable functions of the parameters. We need to propagate the gradient back up until the first module that has parameters that we are interested in optimizing. This implies that all modules following first module that has parameters we want to optimize must implement functions that are differentiable with respect to the input. These assumptions on differentiability are necessary (and sufficient) for the gradient propagation to work. For the objective functions we use, computational time is dominated by the time to calculate the function and the gradient with respect to the final feature vectors and the extra computational effort to propagate the gradients is negligible.

We note that if any of the modules have parameters that are constrained, we can either reparametrize to remove these constraints or restrict the line search during the optimization algorithm to make sure that we always satisfy the constraints.

3. Choice of objective function
Typically speech recognition system parameters are trained/adapted to maximize likelihood. When considering adaptation of features we need to be careful to compare likelihoods in the same feature space. To do this we need that the new features \( x' \) are given by \( x' = h(x) \) where \( h \) is an invertible function and \( x \) represents the original features. In this case we can use the negative of the log likelihood as the objective function to minimize:
\[
g_{\text{ML}}(h(x)) = -\log \det \frac{dh(x)}{dx} - \log P(h(x)|M,G)
\]
The graph \( G \) is a graph that describes the set of word sequences that are allowed. The graph \( G \) can be taken to be either consisting of only the true word sequence (if it is known) or the entire decoding graph if the true word sequence is not known.

In the case that the adapted features are not an invertible function of the original features we cannot use the likelihood as an objective function. This may occur if any of the modules is a non-invertible function. In this case, the negative of the MMI objective function may be more appropriate:
\[
g_{\text{MMI}}(x) = \log P(x|M,G_D) - \log P(x|M,G_N).
\]
(3)
In (3) \( G_N \) is the “numerator” graph consisting of only the true word sequence, \( G_D \) is the “denominator” graph consisting of all possible word sequences. Note that in MMI training we have flexibility with regards to the denominator model and graph. We could take \( G_D \) the denominator graph to represent the set of all word sequences allowed in the application or to be some small subset thereof. In one extreme we can consider the competing model to be a single full covariance Gaussian model whose distribution is learnt from the test data. In this case we can explicitly solve for the mean and covariance of the Gaussian and the log likelihood of the denominator becomes \(-N \log \det(T)\) where \( T \) is the covariance of the test data and \( N \) is the number of frames. In this case the objective is
\[
g_{\text{MMI}}(x) = -N \log \det T - \log P(x|M,G_N).
\]
(4)
When we consider \( g_{\text{ML}}(x) \) as the objective function we could use the EM algorithm and derive a auxiliary function that is faster to evaluate than the likelihood. In this paper we do not do this and we optimize the \( g_{\text{ML}}(x) \) directly.
\( g_{\text{ML}}(x) \), \( g_{\text{MMI}}(x) \) and \( g_{\text{MMI}}(x) \) are the three objective functions that we consider in this paper. To calculate the derivatives of these three objectives w.r.t to \( x_t \), we now write \( \log P(x|M,G) \) explicitly. The model \( M \) is a hidden Markov model with context dependent states, each state being model with a mixture of Gaussians. Thus we can write:
\[
\log P(x|M,G) = \log \sum_{g^n \in G} P(g^n|x)P(x|g^n),
\]
where \( g^n \) is a sequence of Gaussians, \( G \) is the set of Gaussian sequences determined by the model \( M \) and the graph \( G \). The gradient \( \frac{\partial \log P(x|M,G)}{\partial x_t} \) a given frame \( x_t \) is given by:
\[
\begin{align*}
\frac{d \log P(x|M,G)}{dx_t} & = d \log \sum_{g^n \in G} P(g^n|x)P(x|g^n) \\
& = \sum_{g^n \in G} \frac{P(g^n)P(x|g^n)}{P(x)}dg^n \\
& = \sum_{g \in \mathcal{G}_t} \gamma(g_t) \frac{d \log P(x_t|g)}{dx_t} \\
& = \sum_{g \in \mathcal{G}_t} \gamma(g_t) \Sigma_g^{-1}(\mu_g - x_t).
\end{align*}
\]
We see that we can calculate both \( \log P(x|M,G) \) its gradient w.r.t \( x_t \) using the forward-backward algorithm. This means that for both \( g_{\text{MMI}}(x) \) and \( g_{\text{MMI}}(x) \) we can calculate the function and gradient using the forward-backward algorithm.

One remaining piece to calculate the gradients for our objective functions is \( \log \det T \). It’s gradient w.r.t to a frame \( x_t \) is given by:
\[
2T^{-1}(x_t - \mu),
\]
where \( \mu \) is the mean of the data.

Using the gradient of the objective function with respect to \( x_t \) we can calculate the gradient of the objective function w.r.t all of the front-end parameters of interest using the chain rule as discussed in Section 2.

4. Gradients for common front end modules
In this section we break the standard speech recognition front end into modules and give the gradients of each module w.r.t the input and w.r.t the modules parameters (if any). The front end can be broken into the following modules: 1) Windowing 2) Magnitude of the Spectrum (FFT) 3) Mel binning 4) Non-linearity (usually a logarithm) 5) Discrete cosine transform (DCT) possibly keeping only top few coefficients 6) Feature normalization 7) Splicing of frames followed by applying LDA or matrix representing the feature, derivative and double derivatives.

\[
\frac{d}{dx_t}
\]

In all of our experiments we do not consider any parameters before obtaining the magnitude of the spectrum. Thus we only need to consider modules after step (2). We now explicitly give the calculations we need to do in the gradient propagation step given in Equations 1 and 2 for various modules.

For frame by frame linear modules $y_i = f_i(y_{i-1}, A) = Ay_{i-1}$, we have

$$\frac{dg}{dy_{i-1}} = A \frac{dg}{dy_i},$$

and

$$\frac{dg}{dA} = \frac{dg}{dy_{i-1}} y_{i-1}^T.$$

This applies to the Mel binning, DCT and the LDA steps.

For feature normalization by mean subtraction there are no parameters in the module and the gradient can be propagated using:

$$\frac{dg}{dy_{i-1}} = \frac{dg}{dy_i} \cdot \text{mean}(\frac{dg}{dy_i}),$$

where the mean is over frames in the particular utterance under consideration.

If we use the logarithm in step (4) the gradient w.r.t the input is

$$\frac{dg}{dy_{i-1}} = \frac{dg}{dy_i} \cdot /y_{i-1},$$

where `/` denotes component-wise division as in Matlab.

If we use a power compression law in step (4) with $\alpha$ being the power in dimension $j$ then

$$\frac{dg}{dy_{i-1}} = \alpha \frac{dg}{dy_i} \cdot y_{i-1}^{\alpha-1},$$

and

$$\frac{dg}{d\alpha} = \frac{dg}{dy_i} \cdot y_{i-1}^{\alpha-1} \cdot \log y_{i-1}.$$

5. Experimental setup and results

The experiments reported on in this paper were performed on an IBM internal database [10]. The test data consists of utterances recorded in a car at three different speeds: idling, 30 mph and 60 mph. Four tasks are included in the test set: addresses, digits, commands and radio control.

Training data was also collected in a car at three speeds. Since most of the data was collected in a stationary car, the training data was augmented by adding noise collected in a car to the data collected in a stationary car. Data was collected with microphones in three different positions: rear-view mirror, visor and seat belt. The database used for training consisted of 462388 utterances. The baseline acoustic model was word internal with 680 states and 10253 Gaussians.

Most of our experiments were done in an enrollment adaptation scenario. Each speaker provided roughly 300 utterances of data with equal amounts in idling, 30mph, and 60mph conditions. We use 25 utterances of data provided when the car was idling as enrollment data which we use to adapt the front-end parameters. Since the enrollment data was only collected in clean conditions, we add to this noise collected in a car to simulate some of the testing conditions. We decode the remainder of the test utterances with the adapted system. The test set contained 64300 words.

We conducted experiments with two systems. The first was a SPAM [11] system where the inverse covariances are modeled in a linear subspace. The front end for this system consisted of cepstra where all but $c_0$ (the energy) are mean normalized and the energy is normalized by subtracting the maximum energy for that utterance. Nine frames of the cepstra are then spliced and projected using an LDA matrix to 52 dimensions.

Due to the fact that the max function is not differentiable everywhere we also considered a second system where the normalization of all dimensions is done by subtraction of the mean. Also in this system the compression function used after the Mel binning (module 4 of the front end) was a power function with the power being the seventh root. This allows us to consider adapting of the power. The final features were obtained using the static features and their derivatives and second derivatives calculated using nine frames. The acoustic model in this case was a standard mixture of diagonal Gaussians.

We first compared the optimization method used here to the standard technique for optimizing a feature space transform. In this experiment we perform unsupervised adaptation on all the data of a speaker and then re-decode the same data. The baseline error rate was 1.99%, using the standard technique fixing the word scripts from the baseline decode we get an error rate of 1.38%. Directly optimizing the likelihood we get an error rate of 1.30%. This shows that directly optimizing the likelihood is feasible and works better than the standard technique.

All experiments we report on from this point on are in the enrollment scenario. Since we will have to resort to using $g_{MMI}$ as the objective function as we move further back in the front end chain, we evaluate how discriminative training performs for linear transforms. Using $g_{MMI}$ as the objective function to estimate a linear transform we obtain a WER of 1.74%. Although this is an improvement over the baseline it is not as good as using $g_{ML}$ which gives us a WER of 1.55%. Comparing WER’s on the set on which the transforms were trained (this set has 5195 words) we see that using $g_{ML}$ gives us an error rate of 0.52% where as $g_{MMI}$ gives us an error rate of 0.04%. Thus even though using MMI eliminates almost all the errors on the training set its performance does not generalize to the test set. Using an interpolation of the two objective functions the best WER we obtained was 1.49 (with the weight for $g_{MMI}$ being 0.975). While estimating feature transforms discriminatively has been reported upon before [12], the straightforward method we have here of directly optimizing the objective function is novel and avoids the auxiliary function method, the correctness of which is hard to prove.

Having established that the basic framework works by testing it on linear transforms of the features we now report on experiments where we adapt other parameters in the front end. First we consider the LDA projection matrix. Adaptation of this matrix has been considered in the literature [13]. Since the LDA matrix is a projection we cannot consider $g_{ML}$ as our objective function. We use $g_{MMI}$ (WER 1.76%) and $g'_{MMI}$ (WER 1.59%) instead. We do not improve upon the case where we just adapt just a final linear transform, moreover we have more parameters to estimate in the LDA matrix. We next consider adapting a linear transform before the splicing operation (Step 7) in the front end chain. Using $g'_{MMI}$ (WER 1.56) gives us results almost as good as a linear transform of the final features. This transform before splicing is of size 13x13 and has sixteen times fewer parameters than a linear transform on the final feature space. Estimating the transform using $g_{MMI}$ (WER 2.22%) performs poorly and performs worse than the baseline.
We note once again that on the data that the transform was estimated we perform very well (WER 0.04%) when we estimate the transform discriminatively, but this improvement does not generalize to the remaining data from the speaker. It is surprising that the degradation seems to be worse in the case of a transform with fewer parameters. Using a transform before the LDA has the advantage of having many fewer parameters, so it should require smaller amounts of data to estimate. Table 1 compares the performance of linear transforms before and after the LDA with varying amounts of adaptation data. We see that with small amounts of adaptation data the transform before the LDA outperforms the transform after, even if we restrict it to be block diagonal. In the table bd-13 refers to a block diagonal linear transform with four 13x13 blocks.

Finally we note that adapting linear transform both before and after the LDA we get marginally better performance (WER 1.52%) than either transform alone.

The module before the splicing is normalization, which is not differentiable everywhere if we use the max to normalize the energy dimension. So we switch attention to System 2, which uses mean normalization for all dimensions and the seventh root for compression. Table 2 reports our results with this system. We note that in these cases we get significant improvements by adapting the DCT matrix, or a linear transform before or after the lda matrix. In fact in this case we get the best performance adapting a linear transform before the splicing operation. Adapting the power in the compression or the Mel bins doesn’t seem to give further gains.

Table 1: Comparisons of linear transforms before and after the LDA with varying amounts of data

<table>
<thead>
<tr>
<th>Trns. type</th>
<th>num. parms</th>
<th>Adaptation sents.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pre-LDA</td>
<td>169</td>
<td>1.87 1.61 1.57 1.56</td>
</tr>
<tr>
<td>Post-Lda</td>
<td>2704</td>
<td>2.08 1.67 1.60 1.55</td>
</tr>
<tr>
<td>Post-Lda bd-13</td>
<td>676</td>
<td>1.87 1.77 1.76 1.74</td>
</tr>
<tr>
<td>Post-Lda bd-4</td>
<td>208</td>
<td>1.88 1.85 1.84 1.85</td>
</tr>
</tbody>
</table>

Table 2: Adaptation at various stages in the front end

<table>
<thead>
<tr>
<th>Trns. type</th>
<th>objective function</th>
<th>WER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>-</td>
<td>2.61</td>
</tr>
<tr>
<td>LinTrans post-Lda</td>
<td>g_{ML}</td>
<td>1.79</td>
</tr>
<tr>
<td>DCT</td>
<td>g_{MMI}</td>
<td>1.68</td>
</tr>
<tr>
<td>DCT + root compression</td>
<td>g_{MMI}</td>
<td>1.69</td>
</tr>
<tr>
<td>LinTrans pre-LDA</td>
<td>g_{MMI}</td>
<td>2.65</td>
</tr>
<tr>
<td>Mel bins</td>
<td>g_{MMI}</td>
<td>2.65</td>
</tr>
</tbody>
</table>

Table: Adaptation at various stages in the front end

Adapting the Mel bins is an extension of VTLN which can be thought of as a constrained change to the Mel bins. Note that usually VTLN is used in an adaptive training framework and that may be something we need to do to get gains.

6. Conclusion

We have presented a flexible framework for adapting the parameters of the speech recognition front-end which generalizes previous work in adapting the features. In particular we optimize a linear transform on the final features to optimize the MMI objective directly. We show that we can obtain similar performance gains to a linear transform of the final feature space by linear transforms before the LDA matrix which has far fewer parameters and can be estimated with smaller amounts of data.

Future work would explore a better objective function, better techniques for optimization for parameters earlier in the front-end chain, and trying these techniques in an adaptive training framework.

7. References